A HYBRID APPROACH TO PROBLEMS IN IMAGE PROCESSING

BY

TIAN XIA

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2010

Urbana, Illinois

Doctoral Committee:

Professor Yizhou Yu, Chair
Professor David Forsyth
Professor Derek Hoiem
Professor Rui Wang, University of Massachusetts at Amherst

Abstract

Digital image processing generally refers to the use of computer algorithms that deal with operations on or analysis of digital images. It covers a wide range of intriguing sub-topics. While these topics may vary from pattern recognition, classification to synthesis and generation, most techniques recognize the input as a raster image by default, treat it as a two-dimensional signal and applying standard signal-processing techniques to it. In this paper, we attempt to solve specific problems of image segmentation, classification and enhancement by a hybrid approach. We perceive an input image not only as a 2D grid of pixels, but also as a connected graph, a collection of heterogeneous data points, or a surface mesh embedded in 3D space depending on the specific target problem. The broader framework makes it possible to combine image processing techniques with techniques that are typically applied to other research areas including digital geometry processing, machine learning, surface modeling and optimization. We demonstrate by experiments and comparisons that these methods, in concert with standard image operations such as filtering or feature detection, work particularly well and yield satisfactory results for the original target problems.

To Father and Mother.

Table of Contents

Chapter 1

Introduction

Digital image processing generally refers to the use of computer algorithms that deal with operations on or analysis of digital images. It covers a wide range of intriguing sub-topics. While these topics may vary from pattern recognition, classification to synthesis and generation, most techniques recognize the input as a raster image by default, treat it as a two-dimensional signal and applying standard signal-processing techniques to it. In this paper, we attempt to solve specific image processing problems by a hybrid approach.

We mainly focus on three problems, image vectorization (enhancement and generation), image segmentation and classification. Image vectorization is the process of converting a raster image to a vector based representation. We notice that a 2D image can be perceived as a surface mesh embedded in 3D space by treating the color channel as the third dimension. Once the embedded mesh is constructed, geometry processing techniques such as simplification or optimization can then be applied on it. Inspired by this transformation, we have successfully developed a patch based vector representation and its associated vectorization algorithm for full-color raster images. The vector representation is achieved through a series of mesh simplification operations and a triangular patch optimization. There are two important characteristics of our representation. First, the image plane is decomposed into nonoverlapping parametric triangular patches with curved boundaries. Such a simplicial layout supports a flexible topology and facilitates adaptive patch distribution. Second, a subset of the curved patch boundaries are dedicated to faithfully representing curvilinear features. They are automatically aligned with the features. Because of this, patches are expected to have moderate internal variations that can be well approximated using smooth functions.

We have developed effective techniques for patch boundary optimization and patch color fitting to accurately and compactly approximate raster images with both smooth variations and curvilinear features. A real-time GPU-accelerated parallel algorithm based on recursive patch subdivision has also been developed for rasterizing a vectorized image. Experiments and comparisons indicate our image vectorization algorithm achieves a more accurate and compact vector-based representation than existing ones do.

In addition to the standard image processing operations, we employ techniques from other areas such as surface approximation and geometry processing. Digital geometry processing is an intriguing topic in computer graphics itself. Mesh processing techniques such as mesh simplification or mesh quality optimization are very active research areas lately. Mesh quality optimization, for example, plays a central role in many important applications, ranging from computational simulation of physical phenomena to generating visual effects for entertainment. Typically, computational simulations rely on high-quality meshes to model physical objects. Meshes with badly shaped elements degrade both the accuracy and efficiency of the simulation. Traditionally, mesh optimization has relied on global algorithms which are ill-suited to the massive meshes demanded by many modern applications. In this paper, we describe a streaming framework for tetrahedral mesh optimization. We provide empirical results demonstrating that streaming is faster and more memory efficient than global optimization while resulting in essentially identical mesh quality. We also describe a novel, local method for optimizing the surface of a tetrahedral mesh that is efficient, preserves features, and significantly increases mesh quality.

These projects are fully automatic and do not involve any user interaction. However, some image processing problems are not able to be solved in a fully automatic manner. Selecting desired textures and textured objects from an image or video, for example, is a challenging problem in image/video editing. User interaction is often required to achieve the satisfactory result, and how to minimize the manual work done by the user is of significant importance. We present a scalable framework that accurately selects textured objects with

only moderate user interaction. Our method applies the active learning methodology, and the user only needs to label minimal initial training data and subsequent query data. An active learning algorithm uses these labeled data to obtain an initial classifier and iteratively improves it until its performance becomes satisfactory. A revised graph cut algorithm based on the trained classifier has also been developed to improve the spatial coherence of selected texture regions. We show that our system is responsive even with videos of a large number of frames, and it frees the user from extensive labeling work. A variety of operations, such as color editing, compositing and texture cloning, can be then applied to the selected textures to achieve interesting editing effects.

A variant of this technique finds applications in classification of microscopic medical images. We experiment with a microscopic image database of prostatic gland units, and develop effective segmentation and classification methods for automatic detection of malignant gland units in those cross-sectional microscopic images. Both segmentation and classification methods are based on carefully designed feature descriptors, including color histograms and texton co-occurrence tables.

These projects are all conducted by combining image processing techniques with techniques that are typically applied to other research areas. In this paper, we demonstrate by experiments and comparisons that these methods, in concert with standard image operations such as filtering or feature detection, work particularly well and yield satisfactory results for the original target problems.

Chapter 2

Streaming Mesh Optimization

Before we delve into practical image processing problems, we describe in this chapter a novel mesh simplification and optimization technique that aims to handle a classic problem in digital geometry processing. In particular, we design a streaming framework for volumetric and surface mesh optimization to process mesh data too massive for traditional algorithms. These techniques not only find applications in general mesh processing, but also provide insights into other problem domains such as image enhancement and generation.

2.1 Introduction

Computational simulation is an important component in such diverse activities as medical imaging, engineering design, cinematic special effects. These simulations frequently rely on tetrahedral meshes to model the physical domain of interest. The popularity of tetrahedral meshes stems from their ability to accurately model extremely complex geometries. High resolution tetrahedral meshes are often particularly desirable, as they enable greater simulation accuracy and the ability to model phenomena at previously infeasible scales. Fortunately, the power of modern computing and data acquisition technologies has enabled the production of tetrahedral models of enormous size. Unfortunately, many popular mesh processing methods are impractical for large meshes. The reasons for this vary, but generally the problem arises from assuming efficient random access to the mesh data. When the mesh is too large to fit in memory, this assumption fails and inefficient memory access patterns make processing intractable.

One recently proposed solution to this problem is *mesh streaming*. In their seminal paper on the subject, Isenburg and Lindstrom [1] describe a streaming representation for polygonal surface meshes, mixing the vertex coordinates and connectivity data describing the mesh into a single coherent stream. Applications read the stream and have access to a sliding window over the mesh. By exploiting the coherence of the data stream, algorithms can process meshes of arbitrary size using only a fixed-size memory buffer.

In this project we describe, what is to our knowledge, the first streaming algorithm for tetrahedral mesh optimization. Mesh optimization is a vital operation, as many meshes suffer from poor element quality when they are initially generated. Poorly shaped elements negatively impact simulation accuracy and efficiency. In the worst case, if mesh quality is excessively poor, a simulation will simply be unable to proceed. Mesh optimization seeks to avoid such problems by relocating mesh vertices in such a way as to increase mesh quality.

One contribution of our project is an empirical demonstration that streaming optimization improves on traditional global optimization algorithms. Streaming requires significantly less memory and can be applied to meshes with sizes that are prohibitive for global methods. Streaming optimization is also faster than global optimization. This increased scalability and performance comes at essentially no cost in terms of mesh quality. In our experiments, the final quality of meshes optimized via streaming and via global optimization were virtually identical. Another important contribution is a novel method for optimizing the surface of a tetrahedral mesh. This new method is feature preserving, operates locally, and is fully integrated into the streaming framework. When applied in concert with volume mesh optimization, this surface optimization can generate significantly higher-quality meshes than volume optimization can achieve on its own.

2.2 Related Work

Mesh optimization and stream processing of data are very active research areas and a full review of the literature in these fields is beyond the scope of this paper. In the interest of brevity, we will quickly review only the work in these fields most directly relevant.

2.2.1 Streaming Mesh Processing

As previously mentioned, Isenburg and Lindstrom[1] were the first to propose a streaming representation for polygonal meshes. Streaming has subsequently been applied to a variety of large-scale geometry processing applications. Streaming algorithms are used in compression[2] and simplification [3] of massive polygonal meshes. Streaming has also been applied to polyhedral volume meshes, which are mostly used and produced in scientific applications. Isenburg *et. al* [4] extend their streaming framework for triangular mesh compression to tetrahedral meshes. A streaming simplification algorithm for tetrahedral volume meshes is presented by Vo *et. al* [5].

2.2.2 Mesh Quality Improvement

Mesh quality generally refers to geometric characteristics of the elements in a mesh, such as volume and shape, which impact the efficiency and accuracy of numerical algorithms. Various metrics have been proposed to measure and control the quality of meshes. An excellent overview of mesh quality metrics was written by Shewchuk[6], in which he described the mathematical connections between geometric properties, interpolation errors, and matrix conditioning.

Mesh quality improvement, *i.e.* mesh optimization, aims to improve the quality of the mesh by vertex repositioning or local topology modification. In this paper, we focus on vertex repositioning algorithms that do not alter the mesh topology. Laplacian smoothing [7] is a simple and easy to implement mesh optimization algorithm but has significant limitations

(e.g. it can produce tangled meshes). More sophisticated techniques formulate a numerical optimization problem in which an objective function is defined using a quality metric measuring desired geometric properties. Examples include optimizing an inverse mean ratio metric [8; 9] and the condition number of the Jacobian matrix [10]. Freitag *et. al* [11] developed MESQUITE, a mesh quality improvement toolkit. State-of-the-art optimization algorithms are implemented in the library along with a set of termination criteria to ensure the solution accuracy. Due to its versatility, efficiency, and robustness, we incorporate the Mesquite library into our streaming framework for the optimization procedure.

In the following section, we review the optimization problem and the state-of-the-art techniques. We explore their inherent limitations, and explain how a streaming model can be applied to the optimization problem.

2.3 Volume Mesh Optimization

Suppose we have an unstructured mesh of N elements and M vertices. Let e_n be the n^{th} element, v_m the m^{th} vertex, where $n = 1, 2, \cdots, N$, $m = 1, 2, \cdots, M$. Also, denote by $\mathbf{x}_n \in \mathcal{R}^{d \times |e_n|}$ the matrix of coordinates for e_n, where $|e_n|$ is the number of vertices referenced by e_n, and d is the dimension of the mesh vertex. For a tetrahedral volume mesh, $d = 3$ and $|e_n| = 4$ for $n = 1, 2, \cdots, N$.

2.3.1 The Optimization Problem

A mesh optimization problem has three components: the quality metric, the objective function, and the optimization algorithm. The *quality metric* is a continuous function $q : \mathcal{R}^{d \times |e_n|} \rightarrow \mathcal{R}$ that measures one or more geometric properties of an element. It describes what is considered to be a good element as a function of its vertices' positions. Specifically, $q_n = q(\mathbf{x}_n)$ denotes the quality of element e_n as a function of coordinates $\mathbf{x}_n$ of e_n's vertices.

While the quality metric is element-based and measures the quality of individual mesh elements, an *objective function* is needed to combine these values into one for the domain of the optimization problem. Mathematically, we need a function $f : \mathcal{R}^N \to \mathcal{R}$, where the input is $Q = [q_1, q_2, \cdots, q_n] \in \mathcal{R}^N$, $n = 1, 2, \cdots, N$, *i.e.* a vector of element qualities over the entire mesh. Since each of the quality elements depends on the coordinates of vertices belonging to this element, the vector Q is a function of the coordinates of all vertices $\mathbf{x} \in \mathcal{R}^{|d|\times M}$ in the mesh, *i.e.* $Q = Q(\mathbf{x})$. From the two functions we compose the *objective function* to be

$$F(\mathbf{x}) = f \circ Q(\mathbf{x}) = f(Q(\mathbf{x})) : \mathcal{R}^{d\times M} \to \mathcal{R}$$

Different strategies of combining element metrics can be applied to the template function f. The harmonic mean, *i.e.* $f(Q) = \frac{N}{\sum_{n=1}^{N} \frac{1}{q_n}}$, for example, aims to improve the average quality while $L2$ distance squared, $f(Q) = ||Q||_2^2 = (\sum_{n=1}^{N} |q_n|^2)$, tends to put more weight on improvement of worst elements.

The optimization problem is now formulated as $\min_{\mathbf{x}} F(\mathbf{x})$ subject to the constraint that positions of boundary vertices are fixed. We are to solve this resulting minimization problem (a smaller value for q_n indicates e_n has a higher quality) for an optimal solution $\mathbf{x}^* \in \mathcal{R}^{d\times M}$ using an *optimization algorithm*.

2.3.2 Optimization Algorithms

We need an efficient and robust solver for the large system of equations presented by the optimization problem. Freitag *et. al* [8] conducted a series of experiments to compare the solution accuracy and efficiency of different solvers. They analyzed six state-of-the-art solvers along with two solvers of their own design. Experimental results show that their custom algorithms, feasible Newton and conjugate gradient, outperform others in solving the mesh shape quality optimization problem.

The conjugate gradient algorithm is memory efficient, as it requires only the value of the

objective function and its gradient ∇F. However, it is observed to be much slower than the feasible Newton algorithm, especially for large meshes. Feasible Newton has a super-linear convergence for most problems. It first solves a quadratic approximation of the objective function to obtain a search direction. Our non-linear objective function $F : \mathcal{R}^{d\times M} \rightarrow \mathcal{R}$ has a Taylor expansion around point $\mathbf{x}$

$$F(\mathbf{x}+\delta) = F(\mathbf{x}) + \nabla F(\mathbf{x})\delta + \frac{1}{2}\delta^T \nabla^2 F(\mathbf{x})\delta + ...$$

Each iteration of the algorithm tries to find a descent vector that minimizes the above quadratic approximation

$$\min_{\delta} g(\delta; \mathbf{x}) = F(\mathbf{x}) + \nabla F(\mathbf{x})\delta + \frac{1}{2}\delta^T \nabla^2 F(\mathbf{x})\delta$$

If a quadratic function has a finite minimum, it is reached at the point where the function gradient ∇F is null and the function Hessian $\nabla(\nabla F)$ is positive definite. Therefore, we are looking for a δ such that

$$\nabla g(\delta; \mathbf{x}) = \nabla F(\mathbf{x}) + \nabla^2 F(\mathbf{x})\delta = 0$$

To find a new search direction, the feasible Newton method solves the system

$$\nabla^2 F(\mathbf{x})\delta = -\nabla F(\mathbf{x})$$

by applying the conjugate gradient algebraic solver. It then performs a linesearch along this direction and finds a improved point.

Despite its fast convergence and robustness in finding a solution, the feasible Newton method inevitably suffers a large memory footprint. In order to solve the system above, it needs the value of the objective function, its gradient, and its Hessian. For a mesh with

M vertices, the Hessian is a $dM \times dM$ matrix that would have a prohibitive memory cost. Global optimization becomes impractical when the mesh is very large and the objective function introduces a non-sparse Hessian matrix, which is often the case. Motivated by this observation, we develop a framework that tries to solve this optimization dilemma with a streaming approach.

2.4 Surface Mesh Optimization

Moving the vertices on the surface of a mesh is fraught with peril. In engineering applications, the mesh is considered to be noise-free and any change to the surface induces some error to the computational domain. For this reason, many mesh optimization methods keep the surface vertices locked in place. Unfortunately, doing so significantly limits the ability to improve the quality of surface tetrahedra. Recent work by Jiao [12] on *null-space smoothing* has shown it is possible to optimize surface while limiting the introduced error. Following this lead, we have incorporated a constrained surface optimization procedure in our streaming framework to improve a mesh when the worst elements reside in boundary regions.

The need to be spatially efficient and fit in a streaming framework is a key design consideration in our surface optimization algorithm. Freitag [13] reported good results optimizing two-dimensional triangular meshes by locally optimizing the position of each vertex within its 1-ring neighborhood. Extending this approach to surfaces in three-dimensional space, we solve an independent optimization problem at each vertex, x, with respect to all its incident tetrahedra, $T(x)$. We define the objective function as

$$F(x) = \arg \max_{t \in T(x)} \cos di(t)$$

where $di(t)$ is the minimal dihedral angle in a tetrahedron t. Like Jiao, we constrain the domain of the optimization problem (i.e. the space in which a surface vertex is allowed

to move) to the tangent plane at the vertex x. Specifically, the space is defined as the projection of the 1-ring facets on this tangent plane. To effectively preserve sharp angles and discontinuities on surface, we check the maximal angle spanned by normals of 1-ring facets at a surface vertex. If the angle is larger than a threshold, we assume a feature occurs and skip the optimization at this vertex.

It should be noted that, unlike Jiao's work, we are optimizing the surface vertex to achieve better quality tetrahedra, rather than better quality surface triangles. While it is common in practice, we have seen that employing a surface optimization method that is unaware of the attendant volume mesh leads to suboptimal volume mesh quality. Our algorithm is able to achieve better results by directly solving an optimization problem the relates the surface vertex position to volume mesh quality.

As our chosen objective function is non-smooth, we employ the simplex algorithm of Nelder-Mead as the solver of this nonlinear optimization problem. In the two-dimensional search space, the algorithm constructs a triangle with vertices $p_0(x_0, y_0)$, $p_1(x_0 + d, y_0)$, and $p_2(x_0, y_0 + d)$, i.e. the original vertex and two points with a displacement d in each direction. Through a series of geometrical transformations such as reflection and contraction, this triangle moves within the plane towards the optimal position, where it contracts itself to a vertex. To further accelerate the whole process, the surface optimization is only carried out at vertices where quality is below a certain threshold. In practice, our surface smoothing optimization works very well, resulting in decent quality improvement with neglectable deviation from the original mesh surface.

2.5 Stream Framework

The tetrahedral mesh data on disk is first organized into an I/O efficient layout. Once this out-of-core preprocessing is done, the data is streamed through a fixed-size memory buffer. We chose the Mesquite library to optimize the element shape quality of the data in-core.

The optimized portion of the mesh is then written back in a coherent format.

2.5.1 Mesh Layout

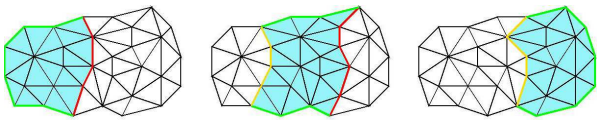
Conventional mesh file formats are not memory-aware. When an element needs to access one of its vertices by its index, the vertices of the entire mesh must be loaded into the main memory, making it infeasible for large meshes. Taking into account of the need for coherence, Isenburg *et. al* [1] propose a streamable format for polygon meshes that interleaves indexed vertices and triangles and provides information when vertices are last referenced.

Similar to the work of Vo *et. al* [5], our streamable format arranges vertices in a coherent order, and then reorders elements with respect to the vertices. In our experience, simple spatial sorting of the vertices along a coordinate axis provides adequate coherence . Our sorting is done using a quicksort variant in conjunction with the Unix system call *mmap*, but any external sorting technique could have been used. When interleaving the reordered vertex and element entries into a new file, we arrange them in a post-order, holding a vertex until all its incident elements are output. As shown in Table 2.1, the the time required to layout very large meshes on disk is quite modest when compared to that required for global optimization, and so should be eminently acceptable in a pre-processing step.

2.5.2 Stream Processing

The complete processing can be broken into five operations. The *read* operation sequentially reads the stream data into a fixed-size memory buffer until the buffer is full. Next, in the *setup* stage, we determine for each vertex that whether it is eligible for optimization. Vertices selected for optimization are then packed and *optimized* by the Mesquite library. This is followed by a pass of surface *smoothing*, in which the optimal positions of surface vertices are determined. After the volume optimization reaches a convergence criterion, all repositioned vertices in the buffer are written back to the output file. The buffer is *updated*, removing data

that has been written out to disk, and the buffer space is recycled. This cycle of operations is repeated until all data has been streamed through the buffer.



(a) (b) (c)

Figure 2.1: Streaming a mesh through a buffer: the shaded part is in buffer. The right-most boundary vertices of the shaded part form the *buffer front*. Vertices on the buffer front in (a) remain in buffer in (b), where they become movable

The streaming process is illustrated in Figure 2.1. There are some subtleties. Vertices are eligible for optimization only when all the 1-ring neighbors of the vertices are also in the buffer. As a result, greater stream coherence yields faster processing as more vertices are optimized within a given buffer. In fact, if the stream is not coherent, it would be possible for the buffer to contain no movable vertices. Faced with such a failure, the only recourse would be to adaptively increase the buffer size. Similarly, buffers can overlap to any degree desired (i.e. some number of vertices and elements are retained from one buffer to another). In our experiments, overlapping buffers to a larger degree had little impact on mesh quality. Given this, the obvious choice is to have the buffers overlap by a single set of tetrahedra which maximizes efficiency.

2.6 Experimental Results

A series of experiments were performed to test the efficiency and solution accuracy of our streaming optimization framework as compared to the global optimization. All experiments were carried out on a machine with an Intel Core 2 Q6600 2.4GHz processor, 4 GB of DDR2 667MHz SDRAM, and a 7200 RPM SATA disk.

2.6.1 Experiment Settings

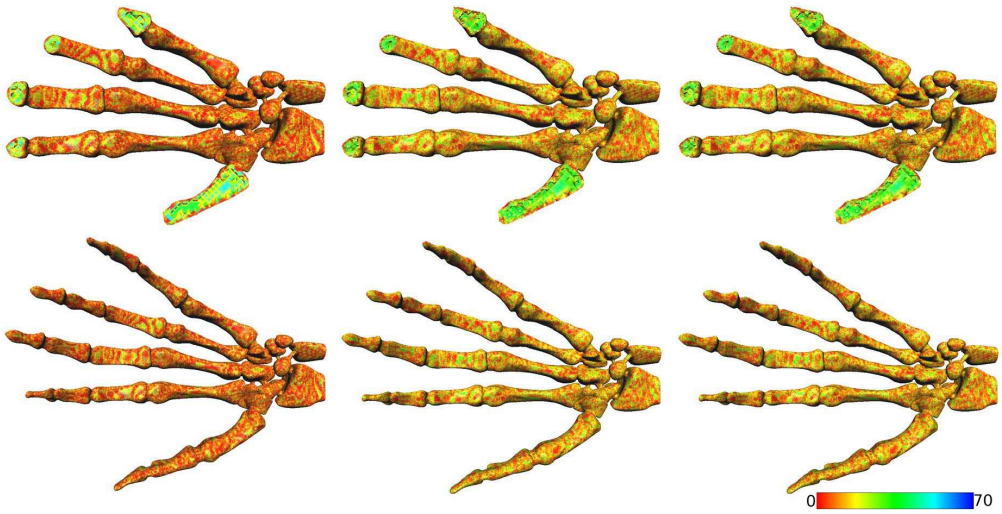
As mentioned in 2.3.1, to formulate an optimization problem we need to define a local quality metric, an objective function template to combine local quality measures into a global one, and an optimization algorithm. We chose the objective function to be the $L2$ distance squared, $\ell_2^2 = (\sum_{n=1}^{N} |q_n|^2)$. This is a reasonable choice as the convex function naturally puts more weight on the improvement of worse elements. The feasible Newton method mentioned in 2.3.2 is selected as the solver. We stop the optimization procedure when the norm of the gradient of the objective function is less than 1.0×10^{-6} (termination criterion).

Another parameter of interest is the choice of the quality metric. In this project, we focus on the shape metrics because they can be used to control important mesh characteristics. The *inverse mean-ratio*, described in [8; 14], is quite representative of the shape metric category and therefore used as the local quality metric for all our experiments. It characterizes the similarity of a tetrahedron as referenced to the ideal equilateral tetrahedron. It is expressed as

$$q_n = \frac{||A(e_n)A(e_R)^{-1}||_F^2}{d \det(A(e_n)A(e_R)^{-1})^{2/d}}$$

where e_n is an arbitrary element in the mesh, e_R is the reference ideal element, and $d(= 3$ in the tetrahedral case) is the dimension. $A(e)$ is the Jacobian matrix of e and $||S||_F = \sqrt{trace(S^T S)}$ is the Frobenius norm. The inverse mean-ratio metric is translation, rotation, and scale invariant. The value of q_n ranges from 1 to $+\infty$. When a tetrahedron e_n has the same shape as the ideal tetrahedron e_R, q_n is 1. The more e_n differs in shape from the ideal element, the larger the value of q_n.

For surface optimization, the step size d in the simplex algorithm is set to one tenth of the projected length of the shortest incident edge. The optimization stops after 15 iterations. To preserve features, we check if the max angle spanned by any pair of normals in 1-ring of a vertex is larger than a certain threshold (normally 75 degree in our experiments). If so, we assume the vertex is on a surface feature indicates a feature and do not reposition it.



(a) Before optimization (b) Global optimization (c) Streaming optimization

Figure 2.2: Dihedral angle visualization of the `hand` mesh

We also optimize only vertices with an incident tetrahedron whose minimal dihedral angle is below 30 degree. This greatly increases the surface optimization efficiency as only around thirty percent of the surface vertices require optimization in our experiments.

2.6.2 Discussion

Table 2.1 summarizes the experimental results. The comparison shows that our streaming algorithm has a significant performance gain over the global optimization. While roughly twice as fast as the global algorithm, the quality tradeoff is almost negligible. Furthermore, streaming is able to process large meshes that do not fit into memory. The Staraft3 model, for example, requires a large amount of memory (more than 4GB) because its complete Hessian matrix needs to be stored in memory. By streaming the data through the buffer, we successfully optimized the mesh using less than 300MB memory. Similarly, the Jet model is simply too large for global optimization, while streaming optimization is able to process it into a usable mesh.

A visual comparison of the relative mesh quality produced by streaming and global optimization can be seen in Figure 2.2 and Figure 2.3. Here, the vertices of the meshes

are colored according to the smallest dihedral angle of a tetrahedron incident upon them. In the case of the Hand model, the poor-quality tetrahedra occur along the surface, likely along features. The vertices in these areas cannot be moved without impacting the feature edges significantly. In situations such as these, any kind of optimization limited to vertex movement alone will fail, and more aggressive processing that changes mesh connectivity must be employed. This is a universal limitation among mesh optimization methods that employ only vertex moevement, not a limitation of streaming. In fact, these images illustrate one of the key findings of our experiments: streaming suffers essentially no limitations in comparison to global optimization. It produces the same quality of mesh more efficiently in terms of speed and memory consumption.

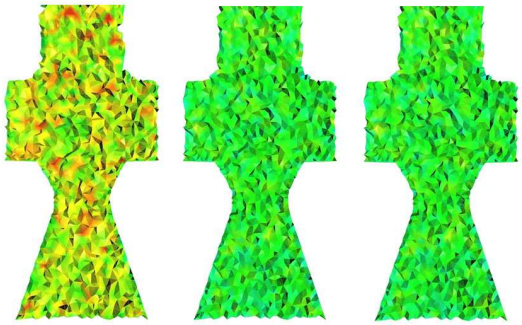
Model	tet count	layout time	max mem. (# iters)	converge time	Inv. Mean Ratio		Dihedral Angle (°)	
					max	mean	min	min 100
staraft1	468,623	—	—	—	26.2609	1.2931	0.4362	2.4772
		—	177M (1)	46s	2.2651	1.1465	12.2053	22.360
		8s	20M (11)	29s	2.2578	1.1468	12.2578	22.078
				61s	1.8964	1.1466	22.2050	25.707
dragon	622,007	—	—	—	197.91	1.7983	0.0395	0.5491
		—	254M (1)	91s	72.067	1.6354	0.1958	1.0235
		10s	20M (18)	50s	72.067	1.6369	0.1958	1.0089
				87s	69.882	1.6370	1.6563	4.5291
hand	1.6M	—	—	—	531473	10.812	1.0×10^{-6}	0.0032
		—	684M (1)	30m47s	4249.3	3.0521	0.0019	0.0597
		41s	40M (27)	8m32s	4249.3	3.0710	0.0019	0.0589
				23m58s	5033.1	2.3525	0.0619	0.1967
staraft2	2.8M	—	—	—	99.1329	1.3191	0.0633	0.7384
		—	1024M (1)	7m11s	3.1930	1.1588	9.7712	14.459
		1m26s	30M (43)	3m32s	3.1930	1.1602	9.7712	14.446
				7m7s	2.6642	1.1588	17.916	22.696
staraft3	14.9M	—	—	—	170.948	1.3220	0.0274	0.5121
		—	out of memory	—	—	—	—	—
		11m25s	187M (33)	27m49s	2.8333	1.1620	11.4105	17.920
				42m46s	3.9242	1.1619	14.3242	20.586
jet	37.8M	—	—	—	99.0725	1.2352	0.0525	0.2666
		—	out of memory	—	—	—	—	—
		22m31s	280M (64)	45m39s	39.7744	1.1041	0.5132	3.7628
				88m55s	14.0867	1.1040	3.5821	9.3119

Table 2.1: Optimization on different models. The four rows of each model show statistics before optimization, using global optimization alone, using streaming optimization alone, and streaming optimization plus surface smoothing, respectively. The converge time and memory usage columns show that our algorithm is typically faster than global optimization while using only a small portion of the main memory. Last four columns are different quality metrics. The value of the inverse mean ratio indicates an element's resemblance to the ideal equilateral element. The closer to 1, the better the quality. A small dihedral angle value implies a flat, *i.e.* badly-shaped element. The last column is the average of the 100 elements with worst quality.

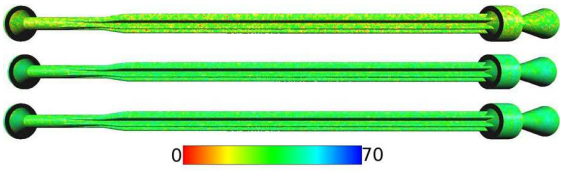
model	staraft1	staraft2	staraft3	dragon	hand	jet
avg edge length	0.0418	0.2086	0.1086	2.3743	118.14	2.1570
max dev.	9.29×10^{-4}	5.36×10^{-3}	3.59×10^{-3}	0.0972	9.495	0.1204
mean dev.	2.15×10^{-6}	4.12×10^{-6}	9.62×10^{-7}	2.46×10^{-3}	0.0896	3.71×10^{-3}

Table 2.2: Deviation of the optimized surface vertices from the original surface. The maximal and mean distance are negligible compare to the average edge length

Using surface optimization in concert with volume optimization proved very effective. The last two columns in Table 2.1 indicate that the minimal dihedral angle in the worst element pushes over 20 degree for model Staraft1. In Hand and Jet, it is more than 6 times larger than that of volume optimization alone. While surface optimization takes as much time as streaming volume optimization, the two methods working together produce meshes of much higher quality than global volume optimization in the same amount of time. As few as two or three passes of surface optimization are sufficient for a substantial improvement. According to our experiments, the quality gain achieved after four passes is only marginal. Figure 2.4 shows the result of adding streaming surface optimization to volume optimization. Table 2.2 details the error induced by our surface optimization method as measured by the distance a vertex has moved off the original surface. For all meshes, the maximum error at a vertex is a small fraction of the average edge length. Moreover, as seen in Figure 2.5, sharp features on the Staraft are very well preserved. Some minor feature erosion can be seen when carefully looking at the conical section between the star grain and the cylinder (specifically, along the circle where the cone

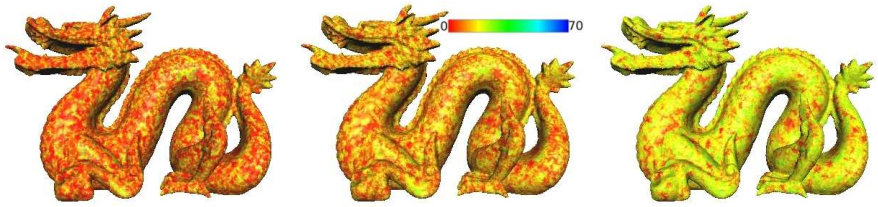


(a) Interior: before optimization, global and streaming optimization



(b) Surface: before optimization, global and streaming optimization

Figure 2.3: Dihedral angle visualization: `staraft`

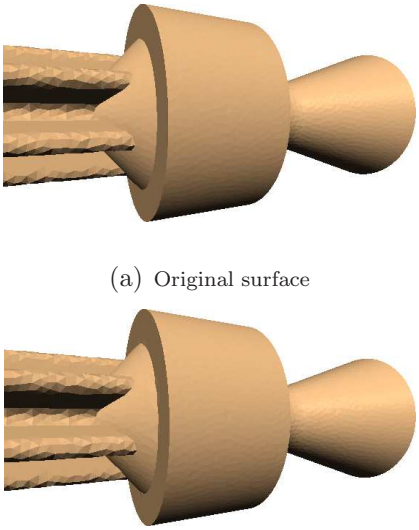


(a) Before optimization (b) Volume optimization (c) Surface optimization

Figure 2.4: Dihedral angle visualization of the `dragon` mesh

and cylinder join). This could be eliminated by tightening the normal threshold parameter, but that would come at the expense of lowered mesh quality.

One should be aware that the angles of the triangles on the surface of the mesh can actually be degraded by our surface optimization method. This is not unexpected, as we are optimizing the tetrahedra that border the surface rather than just the surface. This is, of course, by design; well-shaped tetrahedra are our paramount concern. In our experience, optimizing the surface triangle shape had much less impact on tetrahedra quality, illustrating the benefit of employing a surface optimization method that is aware of the volume mesh.



(a) Original surface

(b) Optimized surface

Figure 2.5: Feature preservation during surface optimization for the `staraft` rocket

2.7 Conclusion

We have described a framework for streaming mesh optimization. In our experiments, streaming has proven superior to traditional global optimization methods in terms of scalability and speed. These gains are achieved at almost no cost. Conversion between traditional mesh formats and a streaming format can be done quickly on relatively

modest hardware. Additionally, in our experiments, streaming mesh optimization exhibited essentially no loss of mesh quality when compared with global optimization. There are several interesting ways in which this work might be extended. We would like to develop a streaming framework for meshes consisting of mixed element types. We would also like to explore other possible streaming layouts of mesh data and their impact on mesh optimization. Finally, both global and streaming mesh optimization are often limited by their inability to move vertices on the mesh boundary. Adding in the capability to perform streaming optimization of the surface mesh would significantly increase the efficacy of our proposed framework.

Chapter 3

Mesh-based Image Vectorization

Under certain circumstances a 2D image can be perceived as a surface triangular mesh embedded in 3D space by using the color information as the third dimension coordinates and treating each pixel as a vertex. Once the image is transformed into a surface mesh, standard mesh processing techniques such as simplification or optimization mentioned in Chapter 2 can be tailored for specific image processing needs. We applied such mesh processing techniques to image vectorization and developed a set of vector representations for raster images.

3.1 Introduction

Vector-based graphical contents have been increasingly adopted in personal computers and on the Internet. This is witnessed by recent desktop operating systems, such as Windows Vista, and multimedia frameworks for rich internet applications, such as Adobe Flash. Needless to say that vector-based drawing tools, such as Adobe Illustrator and CorelDraw, con-

Figure 3.1: MAGNOLIA. Left: original image. Mid left: reconstructed image from our vector-based representation. Mid right: magnification (×4) of the enclosed area in mid left. Right top: magnification of the enclosed area in mid right (original image ×8). Right bottom: magnification (×8) using bicubic interpolation.

tinue to enjoy immense popularity. Such a wide adoption is due to the fact that vector graphics is compact, scalable, editable and easy to animate. Compactness and scalability also make vector graphics well suited for high-definition displays.

Since raster images are likely to remain as the dominant format for raw data acquired by imaging devices, raster image vectorization is going to be of increasing importance. A powerful vector-based image representation and its associated operations need to exhibit the following traits:

• **Representation Power** Vector graphics have been typically used for encoding abstract visual forms such as fonts, charts, maps and cartoon arts. There has been a recent trend to enhance the representation power of vector graphics so that they can more faithfully represent full-color raster images. Such images have regions with smooth color and shading variations as well as curvilinear edges (features) across which there exist rapid color or intensity changes. A primary goal is to design powerful vector primitives that can accurately approximate raster images with as few vector primitives as possible. Since curvilinear features are important visual cues delineating silhouettes and occluding contours, they help users better interpret images. It is essential to use dedicated vector primitives, such as curves, to represent such features. In addition, dedicated primitives can significantly alleviate aliasing along curvilinear features, and make it more convenient to create layers or boundaries by cutting the image open along these features.

• **Automatic Vectorization** From a user's perspective, vectorizing an existing raster image should be hassle free. Almost all the details should be taken care of automatically except for very few tunable parameters.

• **Responsive Rasterization** On the other hand, rasterizing a vector-based image on display devices should not be much slower than opening a regular raster image.

In this paper, we introduce an effective vector-based image representation and its associ-

ated image vectorization algorithm that meets the aforementioned requirements. There are two important characteristics of our representation. First, the image plane is decomposed into a set of nonoverlapping parametric triangular patches with curved boundaries. Such a simplicial layout of patches facilitates adaptive patch distribution and supports a flexible topology with multiple boundaries. Second, a subset of the curved patch boundaries in the decomposition are dedicated to representing curvilinear features. They are automatically aligned with the features. Because of this, patches are expected to have moderate internal variations that can be well approximated using smooth functions. Thus, our vector-based representation can accurately and compactly approximate raster images with both smooth variations and curvilinear features.

We have developed fully automatic algorithms for raster image vectorization and vectorized image rasterization for our vector-based representation. There are a few notable aspects of these algorithms. First, each color channel of the input image is represented geometrically as a triangle mesh. Automatically detected curvilinear features are explicitly represented as boundaries in the mesh by cutting the mesh open along them. Second, a compact vector-based representation needs to have as few patches as possible. We achieve this goal with an adapted feature-preserving mesh simplification algorithm. Mesh simplification is also capable of preserving weak features that have been missed during automatic feature detection and aligning them with patch boundaries. Relying on both feature detection and mesh simplification, our method achieves robust feature alignment with curved patch boundaries. Third, we develop an effective nonlinear optimization method for computing Bézier curves as patch boundaries with respect to an additional constraint that prevents intersections among different Bézier curves. We further perform thin-plate spline fitting to accurately represent the color variations within each patch. Fourth, a real-time GPU-based parallel algorithm is developed for rasterizing vectorized images. It is based on recursive subdivision of the 2D Bézier patches. Experiments and comparisons indicate our image vectorization algorithm can achieve a more accurate and compact vector-based representation than existing ones.

3.2 Related Work

There is a large body of literature on vectorization of non-photographic images [15–17]. These images, often in the form of cartoon arts, maps, engineering and hand drawings, are line- or curve-based and regions in-between curvilinear features are filled with uniform colors or color gradients. Because of this, algorithms are mainly designed for contour tracing, line pattern recognition and curve fitting. A novel representation for random-access rendering of antialiased vector graphics on the GPU has been introduced in [18]. It has the ability to map vector graphics onto arbitrary surfaces, or under arbitrary deformations.

Recent work sees an increasing interest in vectorization of full-color raster images. Compare to line drawings, these images need a more powerful vector representation that accounts for color variations across the image space in addition to curvilinear features. Several softwares, such as VectorEye, Vector Magic, and AutoTrace have been developed for automatic conversion from bitmap to vector graphics. Also available are commercial tools (CorelDRAW, Adobe Live Trace, etc.) that help the user design and edit vector-based images.

Existing vectorization techniques roughly fall into three categories. A few algorithms are based on constrained Delaunay triangulation. For example, [19] developed the ArDeco system for image vectorization and stylizing. It decomposes an image into a set of triangles, and each triangle is filled with pre-integrated gradient. [20] triangulates an image using only pixels on detected edges. Each triangle is then assigned a color by sparse sampling, resulting in blurred, stylish regions. Delaunay triangulation is also used for the image compression algorithm in [21], where an image is approximated using a linear spline over an adapted triangulation. The adaptive approach well preserves features of an image. Overall, these triangulation-based algorithms require a large number of triangles to approximate detailed color variations due to the limited representation power of the color functions defined over each triangle. In addition, each curvilinear feature needs to be approximated by a large number of short line segments. Our technique overcomes these problems and achieves similar

or better reconstruction results with far fewer vector primitives by using more powerful spline patches with curved boundaries.

The second category of techniques aims for a more editable and flexible vector representation. These techniques normally use a higher-order parametric surface mesh. [22] presented an object-based vectorization method, in which a cubic Bézier grid is generated for each selected image object by recursive graph-cut segmentation and an error-driven subdivision. The data fitting capability of cubic Bézier patches prevents a very sparse representation. There are typically many tiny patches in rapidly changing regions. The idea of optimized gradient mesh was introduced in [23]. A gradient mesh consists of a rectangular grid of Ferguson patches. An input image is first segmented into several sub-objects, and a gradient mesh is then optimized to fit each sub-object. The technique in [23] involves manual mesh initialization which aligns mesh boundaries with salient image features. Such user-assisted mesh placement can be time-consuming for an image with a relatively large number of features. The most recent work in [24] proposes an automatic technique to align the boundary of an entire gradient mesh with the boundary of an object layer. However, the rectangular arrangement of patches still imposes unnecessary restrictions to achieve a highly adaptive spatial layout. As a result, it is still challenging to automatically align internal patch boundaries with detailed features inside the object layer. In comparison, our technique automatically aligns patch boundaries with all curvilinear features. In addition, a triangular decomposition offers a flexible simplicial layout that makes it easier to adaptively distribute patches.

Another class of techniques adopts a mesh-free representation and differs from the first two categories in that it treats the color variation as a propagation from detected curvilinear features. Diffusion curves, proposed in [25], creates curves with color and blur attributes, and models the color variation as a diffusion from these curves by solving a Poisson equation. This technique is particularly well suited for interactive drawing of full-color graphics and is a significant leap from other vector-based drawing tools. However, it has limitations when

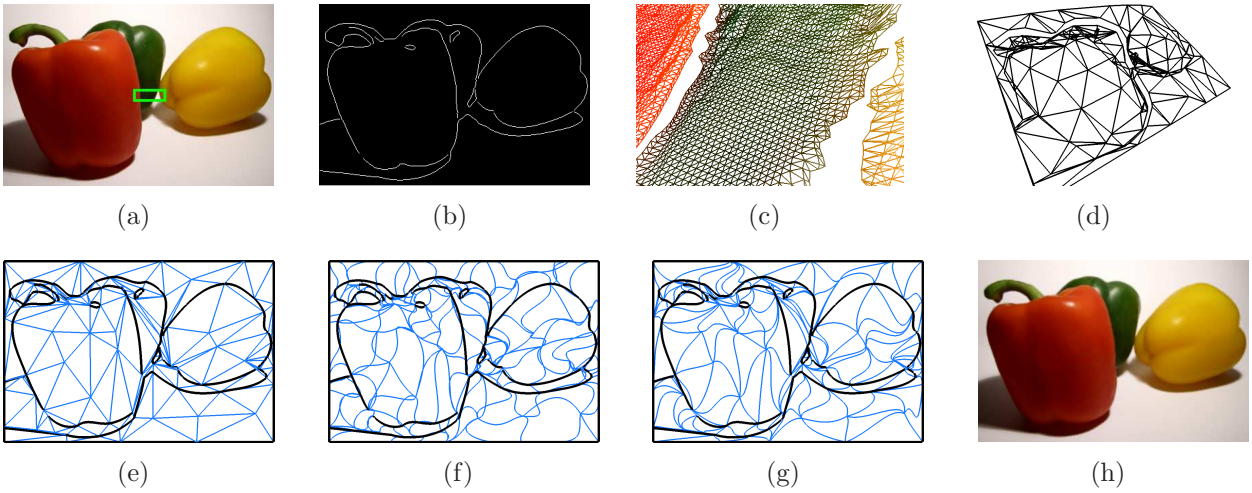


Figure 3.2: Vectorization pipeline. (a) Original image. (b) Automatically detected curvilinear features. (c) Part of the triangular surface mesh constructed from (a) and (b), corresponding to the enclosed part in (a). (d) The coarsened mesh. (e) 2D projection of (d), served as the base domain. (f) The set of initial 2D triangular Bézier patches. (g) Non-overlapping patches optimized from (f). (h) The final vector-based reconstruction by a thin-plate spline color fitting on each patch.

vectorizing raster images. First, it is well known that any solution of the Poisson equation performs membrane interpolation while color variations in raster images may not satisfy this condition especially in regions with relatively sparse features. Second, automatically detected edges have undesirable spatial distributions. While some regions may have overly dense edges, others may have too few edges to accurately approximate the original image. Third, edge-based representation does not guarantee closed regions, making it infeasible to perform region-based color or shape editing. In comparison, our technique is less dependent on edge detection and fits a thin-plate spline to pixel colors within a patch to achieve a more faithful reconstruction.

3.3 Vector-based Image Representation

In our vector-based representation, the entire image plane is decomposed into a set of non-overlapping triangular regions with curved boundaries. We model every triangular region as a 2D triangular Bézier patch. Note that any boundary of a 2D triangular Bézier patch is a 2D

Bézier curve. Every color channel over each 2D Bézier patch is represented separately as a scalar function using a distinct analytical formulation. We chose to adopt thin-plate splines to represent these color channels. The thin-plate splines are defined over the parametric domain of their corresponding Bézier patch.

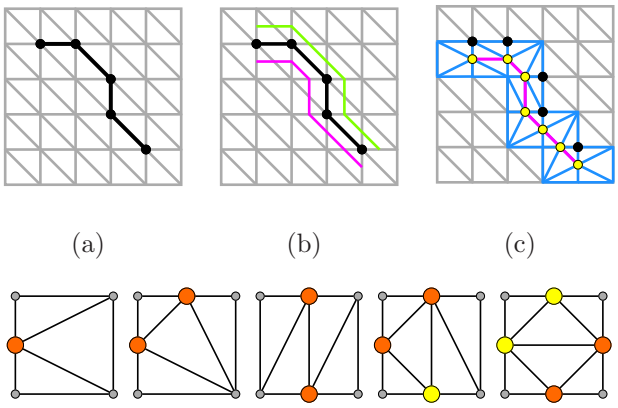
Curved boundaries of 2D Bézier patches lend significant modeling power to the vector-based representation. Compare to straight line segments used in existing triangulation-based approaches, its advantage is twofold: a smaller number of patches and a more efficient color fitting within each patch. When directly applying constrained Delaunay triangulation, a considerably larger number of line segments are needed to approximate a curvilinear feature, resulting in a larger number of triangles. Our method can approximate the same feature at the same precision with much fewer curved segments. Since color discontinuities are detected as features and automatically aligned with patch boundaries in our method, smoother gradients and transitions are left inside patches. Exempt from approximating high frequency information, our patch-wise color fitting scheme *i.e.* thin-plate splines, proves to be very effective. In the absence of a feature at a patch boundary, the two neighboring patches sharing the boundary have a continuous color transition across the boundary.

3.4 Image Vectorization Based on Triangular Decomposition

This section describes our image vectorization technique, and we choose the RGB color space for the vectorization in this paper. For a full-color raster image, every color channel is considered as a height field and converted into a distinct 3D triangle mesh, which we call a *channel mesh*. A feature-preserving mesh simplification is then performed *collectively* on all channel meshes, so their resultant coarse meshes always have the same topology, i.e. the same number of vertices and edges, and project to the same configuration of non-overlapping 2D triangles on the original image plane. The 2D projection of the triangles in the simplified

mesh serve as the *base domain* for Bézier patch formulation and optimization. 2D triangular Bézier patches are computed for every triangle in the base domain. We approximate the color variations over each Bézier patch with a thin-plate spline for every color channel. The main procedure is sketched in Fig. 3.2.

3.4.1 Initial Mesh Construction



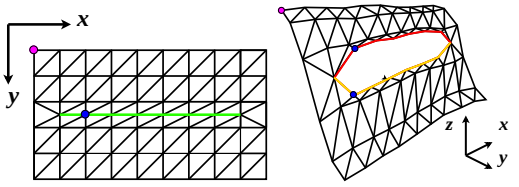
(d) subpixel triangulation patterns. gray dots are pixels. yellow and orange dots are newly inserted subpixels, belonging to different features.

Figure 3.3: (*a*) Triangulation and feature detection at pixel resolution: black dots are pixels on a detected feature. (*b*) Candidates at subpixel resolution: one of the two candidate lines sandwiching the detected feature is where the true discontinuity is. (*c*) Re-triangulation of the affected area (blue tris): new subpixels (yellow dots) along the selected candidate are inserted. (*d*) Re-triangulartion patterns.

An image is first triangulated at pixel resolution on the 2D image plane. Pixels are connected row- and column-wise resulting in an image grid. Every rectangular cell in the grid is divided into two triangles by either of the two diagonals. Since we would also like to preserve important curvilinear image features as much as possible during vectorization, it is necessary to describe the features at subpixel resolution in the triangulation. We start by delineating important image features *i.e.* color discontinuities with image-space edge detection using the Canny detector [26]. For most of our experiments, we use a low threshold of 0.05 and a high threshold of 0.125. Detected features are thinned to 1-pixel wide, linked and removed of T-junctions [27]. Features shorter than 10 pixels are discarded. Since a color discontinuity is associated with at least two neighboring pixels with very different color intensities, a detected feature pixel could be located either at the high end or low end of the disconti-

nuity, and a true subpixel feature should be somewhere between the two extremes. In other words, the true subpixel feature could be on either side of the detected feature as shown in Fig. 3.3(b). We compare image gradients on both sides of the detected feature and, on the side with the larger gradient, a true feature edge at subpixel resolution is traced half-pixel away from the detected feature (Fig. 3.3(c)). The process of inserting subpixels is similar to that of the marching cubes algorithm [28]. Once a new subpixel is created in the middle of a triangle edge, any triangle affected by such edge subdivision is re-triangulated (Fig. 3.3(d)).

Once we have a 2D triangulation with both pixels and subpixels, we elevate it to a triangular surface mesh, called a *channel mesh*, for every color channel. Every pixel in the image space is lifted to a 3D vertex on the mesh. Image coordinates of the pixel is interpreted as the x- and y-coordinates, and the color value at the pixel as the z-coordinate. Every subpixel, however, is *split* into a pair of disconnected *dual vertices*, which have the same x and y coordinates but are assigned different z coordinates. As depicted in Fig. 3.4, the split virtually transforms a subpixel feature in 2D triangulation into a pair of dual features, each representing one side of the original feature. Reflected in the channel mesh, the split cuts the mesh open and leaves a hole in the mesh for every feature. Each dual vertex of the pair carries different attributes associated with one side of the feature. More precisely, each dual vertex gets half of the original connectivity and is assigned the color of its nearest pixel on the same side. To smooth out noises, an optional fairing operation [29] on the z-coordinate can be performed within a narrow neighborhood (normally 1-2 pixels) of dual features.



(a) triangulated 2D image plane (b) channel mesh

Figure 3.4: A pixel in (a) (magenta dot) is lifted to a vertex in (b). A subpixel in (a) (blue dot) is split into a pair of dual vertices in (b). A subpixel feature in (a) (green line) becomes a pair of dual features in (b) (red and yellow lines), forming a hole in the mesh.

Vertex split is crucial in avoiding undesired blurriness along edges during vectoriza-

tion. It effectively preserves the color discontinuities by converting detected features into *mesh boundaries*, which are refrained from collapsing during base domain computation (Section 3.4.2). Also worth of note is that, throughout the initial mesh construction and base domain computation, we ensure all channel meshes differ *only* in the z coordinate at each corresponding vertex. Their projections on the original image plane always correspond to the same 2D triangulation. This important property guarantees a consistent color estimation for different color channel during patch fitting (Section 3.4.4).

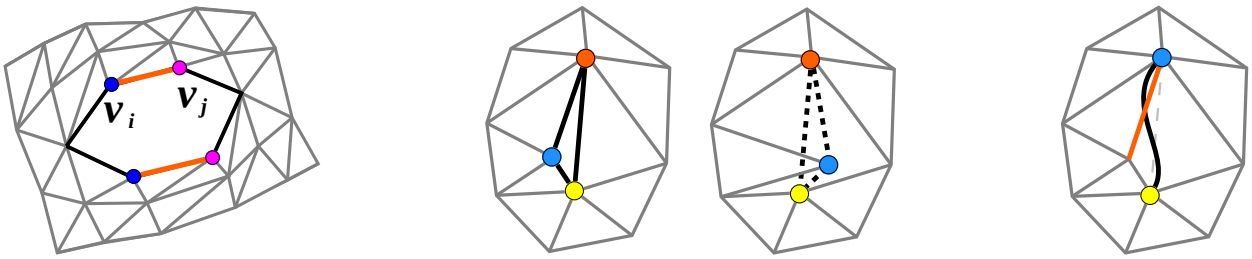
3.4.2 Base Domain Computation

We aim to approximate the entire image with a sparse set of patches, so channel meshes must be simplified to a coarser resolution. Mesh simplification [30; 31] can be thought of as a process that clusters nearby triangles in the original mesh into regions and represent each of the regions using one triangle in the simplified mesh. In the same vein of [31], we apply the highly efficient simplification algorithm based on the quadric error metric. Even though the quadric error metric is not specifically related to thin-plate splines that we use for color fitting, it measures accumulated squared distances to a set of planes and only merges triangles in the same flat region on the mesh. Thus, it is capable of preserving weak features that have been missed during automatic feature detection and aligning them with region boundaries.

Recall that when constructing initial channel meshes, automatically detected features in image space are incorporated as mesh boundaries. We therefore tailor the simplification algorithm as follows.

1. All channel meshes are coarsened in a synchronized manner by applying synchronized edge contractions on all channel meshes. We define the error associated with an edge as the accumulated quadric error over that corresponding edge of all channel meshes.

2. When two boundary vertices contract, their corresponding dual vertices must contract at



(a) dual edge collapse (b) valid vs. flipped (c) intersection

Figure 3.5: (*a*) The pair of orange edges are contracted at the same time. When $\mathbf{v}_i$ (blue dot) collapses into $\mathbf{v}_j$ (the magenta dot), the dual vertex of $\mathbf{v}_i$ (the other blue dot) also collapses into the other magenta dot. (*b*) The right mesh is a flipped configuration of the left one, with a fold-over of the dashed triangle. (*c*) An intersection between a boundary Bézier curve and its immediate neighbor edge, rendering the mesh invalid.

the same time (Fig. 3.5(a)). We define the error of either edge as the average of the pair. This is to assert a seamless projection of the mesh onto the image plane.

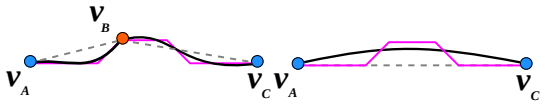
3. Important features should be kept intact during simplification. For an edge contraction, we use one of the edge's two original vertices as the new vertex position. Interior vertices are allowed to collapse both among themselves and into boundary vertices. Boundary vertices, however, are refrained from being absorbed by interior vertices. Furthermore, vertices on the same mesh boundary are allowed to collapse among themselves while contracting to vertices on other boundaries is disabled.

4. At any step during simplification, the 2D projection of any channel mesh should be free of fold-overs. In addition, 2D projections of mesh boundaries are automatically fitted with Bézier curves. And non-boundary edges should not intersect with these Bézier curves when projected onto the image plane. These requirements guarantee the resulting triangular decomposition of the image plane is valid. The use of mesh simplification with additional modifications to generate a triangular base domain is primarily motivated by the fact that constrained Delaunay triangulation (CDT) cannot directly handle Bézier curves as constraints. CDT typically discretizes curves into short line segments.

We impose extra checks when each edge contraction is carried out to enforce the above requirements. After a tentative edge contraction, we project relevant neighborhoods in the

resulting mesh onto the image plane. If a reversed ordering (a flip) is detected in the 1-ring neighborhood of a vertex (Fig. 3.5(b)), we reclaim the contraction, penalize the edge by adding an extra error to the original quadric error, and resume from the next edge with the minimum error. The same measure also applies to planar Bézier curve fitting of subpixel image features. As illustrated in Fig. 3.6, if the edge contraction occurs on mesh boundaries, we re-fit a single new planar curve to the x and y coordinates of the boundary vertices previously represented using two adjacent curves. If any newly fit Bézier curve fails to approximate the original polyline feature segment within a predefined error threshold (our choice is 1 pixel), or it intersects with any of its immediate neighboring edges (Fig. 3.5(c)), we roll back the tentative contraction and restart at the next edge with the smallest error. For better magnification results, control points may be slightly adjusted to enforce G^1 continuity between adjacent curves. In all our experiments, we use planar cubic Bézier curves for boundary fitting.

During simplification we also keep track of vertices being contracted by associating each edge with an *absorbed list*. Consider an arbitrary edge $\mathbf{e}_{ij} = \overline{\mathbf{v}_i\mathbf{v}_j}$ and the 1-ring neighbor of $\mathbf{v}_i$, $N(\mathbf{v}_i)$. Initially, the absorbed list L_{ij} of $\mathbf{e}_{ij}$ consists of its two endpoints, $L_{ij} = \{\mathbf{v}_i, \mathbf{v}_j\}$. As sketched in Fig. 3.7, once e_{ij} is contracted, *i.e.* $\mathbf{v}_i$ collapses into $\mathbf{v}_j$, all other incident edges of the disappearing vertex $\mathbf{v}_i$, $\{e_{ki}, \mathbf{v}_k \in N(\mathbf{v}_i), \mathbf{v}_k \notin N(\mathbf{v}_j), k \neq j\}$, need to be updated by replacing $\mathbf{v}_i$ with $\mathbf{v}_j$ as its new endpoint. $\mathbf{e}_{ki}$ thus becomes $\mathbf{e}_{kj}$, and L_{ij} is appended to L_{ki}, becoming the new absorbed list $L_{kj} := L_{ki} + L_{ij}$. Note that the new list is always ordered such that the first and last entries are always the two endpoints of the edge, $L_{jk} = \{\mathbf{v}_j, \cdots, \mathbf{v}_i, \cdots, \mathbf{v}_k\}$. By the time

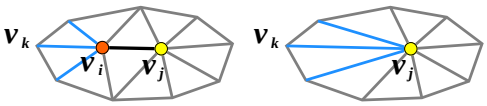


(a) before (b) after

Figure 3.6: The magenta polyline denotes the shape of the original boundary from $\mathbf{v}_A$ to $\mathbf{v}_C$. (a) Before $\mathbf{v}_B$ collapses into $\mathbf{v}_C$, there are three vertices ($\mathbf{v}_A$, $\mathbf{v}_B$, and $\mathbf{v}_C$) on the boundary. It is fitted with two separate Bézier curves, one from $\mathbf{v}_A$ to $\mathbf{v}_B$ and the other from $\mathbf{v}_B$ to $\mathbf{v}_C$. (b) After the collapse the boundary is fitted with a single new curve from $\mathbf{v}_A$ to $\mathbf{v}_C$.

the algorithm terminates, each existing edge will have collected a set of contracted vertices.

Final coarse meshes are projected back to the original image plane. The set of projected triangles on the original image plane is the *base domain* $M^b = (V^b, E^b, F^b)$. For an edge $\mathbf{e}^b \in E^b$ with endpoints $\mathbf{v}_i^b, \mathbf{v}_j^b \in V^b$, there must exist a path connecting these endpoints on the original mesh. Instead of directly searching for such a path, we make use of the ordered vertices in the absorbed list of $\mathbf{e}^b$ to generate an initial path between $\mathbf{v}_i^b$ and $\mathbf{v}_j^b$. This initial path serves as the starting point for Bézier curve optimization. For every base triangle $f^b \in F^b$, we will formulate a triangular Bézier patch.



(a) before contraction (b) after contraction

Figure 3.7: Edge contraction. The edge $\mathbf{e}_{ij} = \overline{\mathbf{v}_i\mathbf{v}_j}$ (black) in (a) is contracted, causing $\mathbf{v}_i$ to collapse into $\mathbf{v}_j$. The three surviving edges (colored in blue) originally incident to $\mathbf{v}_i$ update their absorbed list by appending the absorbed list of $\mathbf{e}_{ij}$ to their own.

3.4.3 Bézier Patch Optimization

The base domain, $M^b = (V^b, E^b, F^b)$, covers the entire image plane. The image will finally be decomposed into a set of non-overlapping triangular Bézier patches. There is a one-to-one correspondence between the base triangles and Bézier patches. For each Bézier patch, the location of its three corners coincides with the location of the three vertices of its corresponding base triangle. Curved patch boundaries, however, are yet to be computed from edges in the base domain. Note that some of these edges correspond to mesh boundaries. Since mesh boundaries are fitted with Bézier curves on the image plane during mesh simplification, these curves are Bézier patch boundaries already and do not need further processing. Other edges, along with their associated traced paths (Fig. 3.8(a)), will be used for computing Bézier patch boundaries.

We compute patch boundaries by formulating it as an optimization problem. Before being used for optimization, the traced path of an edge in the base domain has to be pruned

clean of branches or small loops (Fig. 3.8(b)). Then it is fitted with a Bézier curve that specifies the initial boundary position. These curves, as shown in Fig. 3.8(c), normally intersect with each other, resulting in a set of folding and overlapping Bézier patches. Our goal is to improve upon them to generate a set of non-overlapping patches while boundaries are kept as close as possible to where originally traced paths reside. Inspired by [32], this overall goal is formulated as a nonlinear optimization which is solved iteratively. Within each iteration, we optimize every patch boundary in a sequential order. The optimization of a single patch boundary is elaborated as follows.

Consider a boundary Bézier curve C shared by two triangular patches B_1 and B_2. When optimizing C, we seek the optimal positions of C's control points (excluding two endpoints) as well as control points over the interior of B_1 and B_2 by fixing the control points of the other two boundaries of these patches. If there are intersections between C and other boundary curves of B_1 and B_2, it satisfies that $\exists \mathbf{p}_i \in B_1 \bigcup B_2, |J(\mathbf{p}_i)| \leq 0$ where $J(\mathbf{p}_i)$ is the Jacobian at point $\mathbf{p}_i$ [1]. On the other hand, the Jacobian at any point within a base triangle is always the same and its determinant is always positive. We regard a Bézier patch's corresponding base triangle as its "ideal" reference and try to optimize the determinant of its Jacobian towards that of the base triangle. To make the optimization more tractable, instead of enforcing a positive determinant of the Jacobian everywhere in $B_1 \bigcup B_2$, we try to enforce a positive determinant of the Jacobian at a dense set of sample points in $B_1 \bigcup B_2$. The sample points are uniformly drawn within the parametric domains of B_1 and B_2 as well as on the curve C.

Another factor to integrate into the objective function is the distance between the current boundary C and the initially traced path C^0. We would like to keep C close to C^0. C and C^0 are both uniformly sampled, and the distance is estimated as the sum of distance between

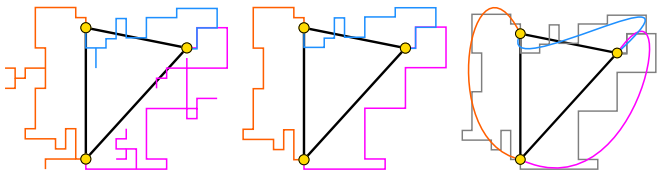
[1] Let $\mathbf{B}(u,v,w) = \begin{bmatrix} x(u,v,w) \\ y(u,v,w) \end{bmatrix}$, where $u+v+w=1$, be a 2D triangular Bézier patch. The Jacobian of $\mathbf{B}$ is defined as a 2x2 matrix, $\begin{bmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} \end{bmatrix}$.

every pair of corresponding sample points. We summarize the objective function as

$$\sum_{k=1,2}\sum_{\mathbf{p}_i\in B_k} w_i(|J_{B_k}(\mathbf{p}_i)| - |J^b_{B_k}|)^2 + \sum_{\substack{\mathbf{p}_j\in C,\\ \mathbf{p}^0_j\in C^0}} w_d \parallel \mathbf{p}_j - \mathbf{p}^0_j \parallel^2, \tag{3.1}$$

where we denote by $J^b_{B_k}$ the Jacobian of the base triangle associated with patch B_k, and $\mathbf{p}_i$ represents a sample point over the patch B_k. $w_i = w_p$ if $|J_{B_k}(\mathbf{p}_i)| > 0$, and $w_i = w_n$ if $|J_{B_k}(\mathbf{p}_i)| \leq 0$. w_p, w_n, and w_d are the weighting factors for Jacobian with a positive determinant, Jacobian with a negative determinant, and the distance, respectively. In all our experiments, $w_p = 0.2$, $w_n = 1$, and $w_d = 5$. As mentioned earlier, the unknowns in the above objective function include C's control points (excluding two endpoints) as well as control points over the interior of B_1 and B_2. Every sample position $\mathbf{p}_i$ is expressed in these unknown control points, and the determinant of the Jacobian is a function with respect to the same set of unknowns.

Since we define two different Jacobians for any sample point on a boundary curve, one for each adjacent patch, the objective function (Eq. 3.1) is C^2 and eligible to be solved by the BFGS algorithm, a quasi-Newton method that approximates the second derivatives of the function using the difference between successive gradient vectors. In practice, we choose the relatively faster implementation bfgs2 in GSL [33]. In all our experiments, we have successfully resolved all boundary inter-



(a) traced paths (b) path pruning (c) initial fitting

Figure 3.8: An initial Bézier patch. (*a*) The base triangle. Every edge is associated with a traced path. (*b*) Traced paths are pruned clean of branches and self loops. (*c*) Pruned paths are fitted with Bézier curves, which may introduce inter-curve intersections and neighboring patch overlap.

(a) features (b) lines (c) feature blurred (d) curves (e) feature preserved

Figure 3.9: Curved boundaries vs. line segments. (a) Part of the object silhouette is not detected due to a weak contrast. (b) Use triangle domain directly. The straight line segments do not align with the object boundary. (c) The result is a blurred boundary region. (d) Boundaries are optimized from traced paths into curves, roughly following the object boundaries. (e) The undetected features are preserved in the result.

sections using 2D cubic Bézier curves as patch boundaries.

The example in Fig. 3.9 demonstrates why curved boundaries are necessary at places with no detected features. Edge detection normally fails at features with a weak color contrast. Fortunately, mesh simplification exhibits a similar behavior with region growing and tends not to grow across such weak features unless there are no other smoother regions left. Thus, when mesh simplification terminates, weak features would follow region boundaries and are often identified as the initial traced paths. Since our patch boundary optimization encourages an optimized boundary curve to follow the initial path, the final optimized boundary is most likely to follow the original weak feature. If we directly took those straight edges from the base domain as the final patch boundaries, a weak curved feature would fall into the interior of a patch and adversely affect color fitting.

3.4.4 Patch Color Fitting

Given a 2D triangular Bézier patch, we need to approximate color variations over the region covered by the patch. Intuitively we would define the color field using the Bézier patch itself, storing a color value as an additional coordinate at every control point. However, low-degree triangular Bézier patches have few internal control points (1 in the case of cubic). Unless a very large number of patches are used, the limited degree of freedom does a poor job in color fitting inside the patch while maintaining continuity across patch boundaries, leading

to severe color distortion. Instead, we apply thin-plate spline fitting in the parametric domain of the patch to achieve the goal. Thin-plate spline (TPS) interpolation [34] is a widely used method for scattered data interpolation. Given the parameters of a set of N points, $\{(u_i, v_i)\}$, in the 2D parametric domain of the patch, each pair of parameters (u_i, v_i) associating with a color value h_i, TPS interpolation attempts to construct a smooth function $f(u, v)$ that satisfies all constraints $f(u_i, v_i) = h_i$. The solution f minimizes the bending energy $I(f) = \int\int_\Omega f_{uu}^2 + 2f_{uv}^2 + f_{vv}^2 dudv$ and is expressed as

$$f(u, v) = \sum_{i=1}^{N} \alpha_i \phi(\| (u_i, v_i) - (u, v) \|) + b_0 + b_1 u + b_2 v, \tag{3.2}$$

where $\sum_{i=1}^{N} \alpha_i = 0$, $\sum_{i=1}^{N} \alpha_i u_i = \sum_{i=1}^{N} \alpha_i v_i = 0$, and (u_i, v_i)'s serve as the center locations of the thin-plate radial basis function,

$$\phi(s) = s^2 \log s. \tag{3.3}$$

Combined with constraints $f(u_i, v_i) = h_i, 1 \leq i \leq N$, one solves for TPS coefficients $\{\alpha_i\}$ by a linear system

$$\begin{bmatrix} \mathbf{K} & \mathbf{P} \\ \mathbf{P}^T & \mathbf{O} \end{bmatrix} \begin{bmatrix} \alpha \\ \mathbf{b} \end{bmatrix} = \begin{bmatrix} \mathbf{h} \\ \mathbf{o} \end{bmatrix}, \tag{3.4}$$

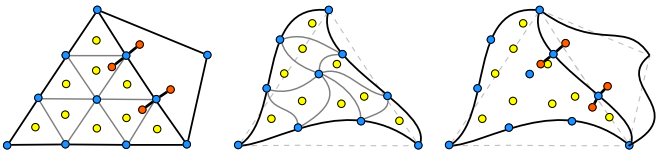
where $K_{ij} = \phi(\| (u_i, v_i) - (u_j, v_j) \|)$, the i-th row of $\mathbf{P}$ is $[1, u_i, v_i]$, $\mathbf{O}$ is a 3×3 zero matrix, $\mathbf{o}$ is a 3×1 zero vector, $\alpha = [\alpha_1, \alpha_2, \cdots, \alpha_N]^T$, $\mathbf{h} = [h_1, h_2, \cdots, h_N]^T$, and $\mathbf{b} = [b_0, b_1, b_2]^T$. Once α and $\mathbf{b}$ are solved, we are able to compute the color, *i.e.* the height value, corresponding to any point in the parametric domain using Equation (3.2).

One drawback of TPS interpolation is that it is sensitive to the constraints. If there is noise in the constraints, the reconstruction result tends to have undesirable undulations. Instead of interpolation, we opt for TPS *fitting* by adding a number of extra constraints

at $\{(u'_i, v'_i)\}$ with color values $\{h'_i\}$. TPS fitting essentially solves for the same number of coefficients in the least squares sense. The formulation of the linear system is almost intact except for the newly added rows. The overdetermined system is written as

$$\begin{bmatrix} \mathbf{K} & \mathbf{P} \\ \mathbf{K'} & \mathbf{P'} \\ \mathbf{P}^T & \mathbf{O} \end{bmatrix} \begin{bmatrix} \alpha \\ \mathbf{b} \end{bmatrix} = \begin{bmatrix} \mathbf{h} \\ \mathbf{h'} \\ \mathbf{o} \end{bmatrix}, \tag{3.5}$$

where $K'_{ij} = \phi(\| (u'_i, v'_i) - (u_j, v_j) \|)$, the i-th row of $\mathbf{P'}$ is $[1, u'_i, v'_i]$, and $\mathbf{h'} = [h'_1, h'_2, \cdots, h'_K]^T$. The fitting version of TPS effectively smoothes out noises and renders a more robust approximation to the given data points.



(a) base triangle (b) Bézier patch (c) inter-patch consistency

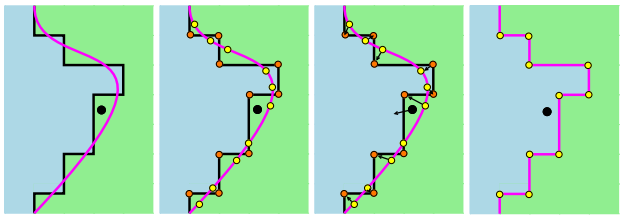
Figure 3.10: (a) In a base triangle, Barycentric coordinates of all centers (yellow dots), corners (blue dots), and additional points (orange dots) sampled near edges, are computed. (b) These coordinates are used for the evaluation of corresponding fitting constraints on the Bézier patch. (c) Additional points (orange dots) serve as constraints for color continuity across patch boundaries.

TPS does not have any restriction on the 2D parametric domain over which it is defined. The color variations over a patch can therefore be accurately approximated if we sample constraints from both interior and edges of the triangular parametric domain. We treat each color channel as a height field over the 2D parametric domain of the Bézier patch. In Fig. 3.10, we illustrate how extra fitting constraints are sampled. We simply take the base triangle as the parametric domain of its corresponding patch. Every edge of the base triangle is evenly divided into n segments, and the base triangle is partitioned into n^2 smaller triangles, where n is typically set to 10. For the center and corners of every subdivided triangle, we compute their barycentric coordinates and evaluate their corresponding points on the Bézier patch. These points on the Bézier patch are actually image-space locations where

color values should be sampled. The color values at any constraint are computed from the input image using bilinear interpolation.

This method may induce missampled color values in regions near detected features. In the image space, a feature is a polyline in subpixel resolution (Fig. 3.3(c)). In the parametric domain, however, the polyline is fitted with a Bézier curve. The polyline and the curve generally intersect with each other. As illustrated in Fig. 3.11(a), a sample point on the left hand side of the Bézier curve is supposed to be blue. But in the image space, it is actually located on the right hand side of the polyline (i.e. the detected feature), and is therefore missampled as a green point.

We resolve this missampling problem with a free-form deformation [35] from the parametric domain to the image domain. As shown in Fig. 3.11(b), for each Bézier curve and its corresponding polyline, we define a set of corresponding point pairs by sampling the curve (arc-length) and the polyline (chord-length). As points on the curve move towards their correspondences on the polyline, the curve eventually deforms into its corresponding polyline (Fig. 3.11(c)). From these feature constraints, a continuous and one-to-one warping from the parametric domain to the image domain is derived. Sample points near Bézier curves will be associated with correct color values once being warped (Fig. 3.11(d)).



(a) (b) (c) (d)

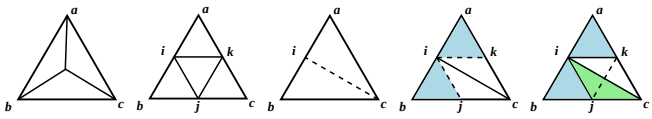
Figure 3.11: Free-form deformation to eliminate missampling.

To guarantee the continuity of derivatives across patch boundaries, we sample additional constraints near the edges of the parametric domain. For every constraint on an edge, we add a pair of extra constraints slightly away from the edge in the direction normal to the edge (Fig. 3.10(a)). Note that patch boundaries that model color discontinuities have only one incident patch, and therefore do not require such additional constraints.

The center locations of the thin-plate basis functions are sampled in the parametric domain in the same manner, only at a much sparser rate. Our experiments show that in the presence of extra fitting constraints, 16 basis functions per patch (with at most 9 bases at the interior of the patch) suffice for a reasonable approximation.

3.5 Vector Image Rasterization

To visualize the vectorized image from the previous section, we need to render the triangular patches and their associated color information onto a discrete image plane. We have developed a real-time GPU-based parallel algorithm for rendering vectorized images. It can achieve more than 30 frames per second on a 768x512 resolution. This algorithm relies on 4-way subdivision of triangular Bézier patches. In the following, we briefly introduce this type of subdivision first, and then present the GPU-based rasterization algorithm.



(a) 3-way (b) 4-way (c) (d) (e)

Figure 3.12: Evaluation of control points by four passes of the de Casteljau's algorithm in a uniform 4-way subdivision. (*c*) 1st pass with $t = (0.5, 0.5, 0)$. (*d*) 2nd and 3rd passes, both with $t = (0.5, 0.5, 0)$. (*e*) Last pass with $t = (1, -1, 1)$.

Conventional 3-way subdivision by the de Casteljau's algorithm does not apply in the case of triangular patch. For example, Fig. 3.12(a) shows a uniform 3-way subdivision with parameters $t = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$, where patch boundaries are never subdivided. Patches only become thinner and thinner but never converge to the desired resolution. Instead, we uniformly subdivide Bézier patch B_{abc} into four smaller Bézier patches B_{aik}, B_{ibj}, B_{kjc}, and B_{ijk}. As illustrated in Fig. 3.12(c) - 3.12(e), we start by partitioning B_{abc} into two halves, B_{aic} and B_{ibc}, using parameters $t = (0.5, 0.5, 0)$ for the de Casteljau's algorithm. We do a similar subdivision on B_{aic} and B_{ibc} respectively. By now, all control points of B_{aik} and B_{ibj} (shaded in blue) are available. The last pass of the de

Casteljau's algorithm is performed on patch B_{ijc} (in green) with parameters $t = (1, -1, 1)$. This gives us the control points for B_{kjc} and B_{ijk}. This last pass performs extrapolation using a negative parameter. Since de Casteljau's algorithm only performs reparameterization without altering the underlying geometry, control points resulting from such extrapolation can exactly reproduce the geometry of the original subpatches B_{kjc} and B_{ijk}.

Our GPU-based rasterization algorithm was developed using NVidia CUDA [36] on a GeForce 8800 GTX GPU. The algorithm has three major stages. First, recursively perform 4-way subdivision (Fig. 3.12(b)) on every original 2D triangular Bézier patch B_i until the bounding box of each resulting patch becomes smaller than a pixel. This stage is highly parallelizable because the subdivision of every patch is independent of each other [37]. We perform in parallel one level of subdivision across all patches that need to be further subdivided. Each thread block has 64 threads and processes 16 patches in parallel because of the limited size of the shared memory. Thus, every 4 threads in the same block perform the 4-way subdivision of a single patch in parallel. Such parallel subdivision is repeated until all resulting patches have become sufficiently small.

The second stage involves three substeps. i) approximate every resulting small patch F_j from the previous stage using the triangle T_j formed by its three corners and compute the 2D bounding box of T_j; ii) for every pixel inside the bounding box, perform point-in-triangle test using barycentric coordinates to identify the pixels covered by T_j; iii) for every pixel covered by T_j, compute its parametric values with respect to patch B_i using the barycentric coordinates of the pixel within T_j and the parametric values of T_j's vertices with respect to B_i. By the end of this stage, we will have figured out for every pixel which original 2D Bézier patch covers it. We parallelize the second stage over the set of finest patches from the previous stage. Each thread block has 256 threads each of which is responsible for executing all three substeps on one distinct patch.

In the third and last stage, for every pixel, we retrieve the thin-plate spline computed for the 2D Bézier patch that covers the pixel, and substitute the parameters of the pixel into

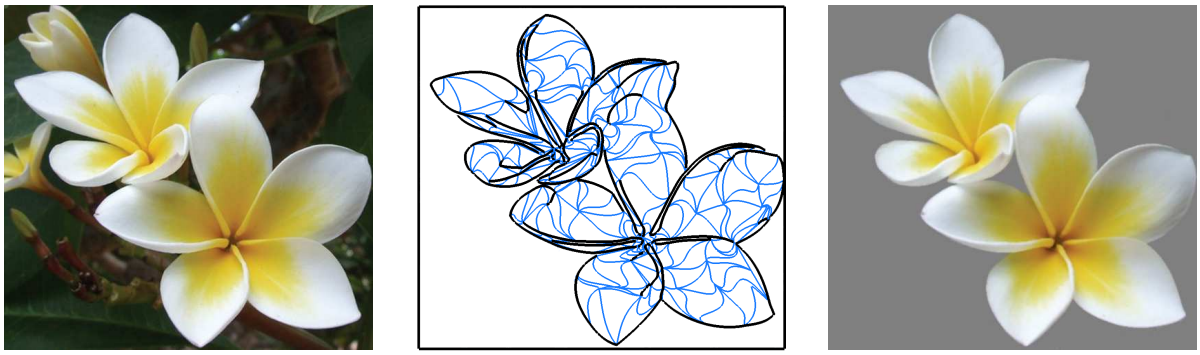


Figure 3.13: Left: original image. Middle: 380 triangular Bézier patches for flower petals. Right: reconstructed result by our representation with 0.98 per pixel mean reconstruction error.

Equation (3.2) to obtain its color. We parallelize this last step over the set of pixels. Each thread block has 512 threads each of which takes care of thin-plate spline evaluation for one distinct pixel.

3.6 Results and Discussions

We have fully implemented our image vectorization algorithm and successfully tested it on a large number of raster images. Thanks to feature-preserving mesh simplification and thin-plate spline fitting, our algorithm is not sensitive to edge detection results, and can achieve high-quality reconstruction with relatively few perceptually important curvilinear features.

Image vectorization results from our algorithm can be found in Figs. 3, 3.13-3.19 and the supplemental materials. We adopt the same parameter setting in most of our experiments and have discussed them in various subsections in Section 3.4. The parameters for Canny edge detection were given at the beginning of Section 3.4.1. The weights for patch boundary optimization were given after Eq. 3.1 in Section 3.4.3. And the sampling scheme of the constraints for thin-plate spline fitting has been discussed at the end of Section 3.4.4. Every Bézier patch uses at most 19 3D TPS terms for color variations and an amortized cost of 4.5 2D control points for its geometry, summing up to 66 scalar coefficients for a complete representation. The number of triangular patches varies from image to image and has been

summarized in Table 3.1 along with mean reconstruction errors. The number of patches to achieve a similar level of reconstruction accuracy increases with the complexity of edge structures in the raster image. Our algorithm typically takes 1-2 minutes on an Intel Core 2 Duo 3.0GHz processor to vectorize a 512x512 image. Our real-time GPU-based rasterization algorithm achieves 60 frames per second on a 384x512 resolution and 32 frames per second on a 768x512 resolution using a GeForce 8800 GTX GPU.

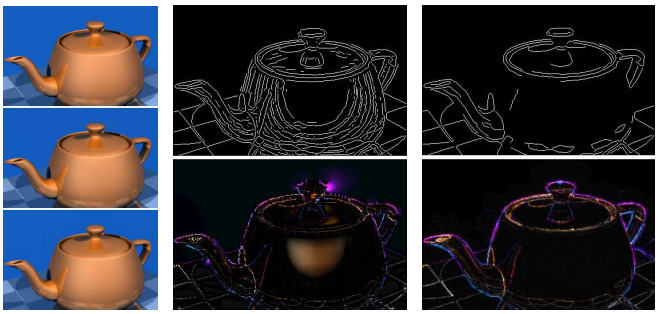


Figure 3.14: Comparison with diffusion curves [25]. Left column: (from top down) original image, reconstruction result using diffusion curves, and using our method, both achieving a reconstruction error around 1.0 per pixel. Mid column: automatically detected feature edges (top) and reconstruction error (amplified by 4) (bottom) of diffusion curves. Right column: feature edges and reconstruction error (amplified by 4) of our method.

We have compared our vectorization algorithm with three automatic ones in the literature [19; 24; 25]. Fig. 3.13 shows our image-space triangular decomposition and vector-based reconstruction of a foreground layer used in [24] (Fig. 9). Compared with their automatic gradient mesh generation technique, our automatic method still achieves the same level of reconstruction quality with far fewer patches and a comparable total degree of freedom. We consider a small number of patches as "being compact". When the number of patches decreases, each patch inevitably covers a larger region of the image and needs a more complex model to approximate its color variations. This tradeoff is indeed necessary because a smaller number of patches better support standard applications such as editing, where fewer patches mean less user manipulation.

The technique in [24] only aligns the boundary of the gradient mesh with the outmost boundary of the foreground layer. It is hard to perform detailed feature alignment within the foreground layer using gradient meshes. As a result, curvilinear features separating petals of

different flowers are not well preserved, which introduces visual artifacts. In comparison, our method easily performs such internal feature alignment and achieves better visual quality. A side-by-side comparison and a cost analysis can be found in the supplemental materials.

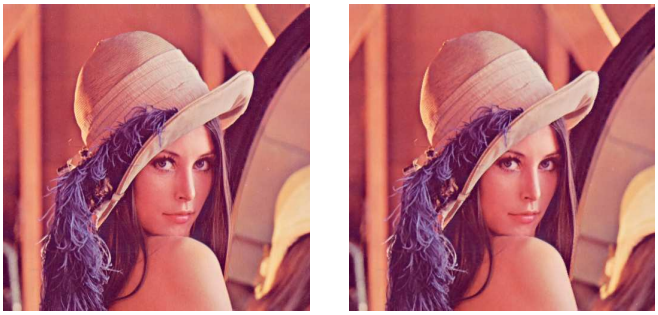


Figure 3.15: Left: Original Image. Right: Reconstructed image by our representation. We achieve a mean reconstruction error = 2.4 with 2011 patches.

Fig. 3.14 shows a comparison with diffusion curves [25]. Our method requires significantly fewer edge features to achieve the same level of mean reconstruction error. Diffusion curves require more edges and therefore have to lower edge detection thresholds. At low thresholds, the detected edges have an undesirable spatial distribution. Some smooth regions are filled with overly dense edges while others have insufficient number of edges, such as the central highlight area in the teapot image. Because shading variations in these undersampled areas do not closely follow membrane interpolation, color approximation using diffusion gives rise to relatively large errors. Because of triangular decomposition and thin-plate spline fitting, our method achieves better reconstruction quality in regions with insufficient number of edges. To achieve the same level of reconstruction error, diffusion curves require $36.46K$ coefficients in total (Bézier curves for edge representation and polyline fitting for colors and blur values) while our representation has $66 \times 525 = 34.65K$ degrees of freedom. Fig. 3.15 shows our vector-based reconstruction of Lena. Our method successfully preserves a large number of tiny details and achieves far better visual quality than the ArDeco system (Fig. 6 in [19]).

In addition to successful reconstructions at the original resolution, our representation also makes it easy to perform standard vector graphics operations such as magnification or editing. Our vectorization is a lossy procedure because thin-plate spline fitting removes details

with extremely high frequencies.It should therefore be considered as a type of image stylization, where important image features are preserved while certain internal high-frequency variations cannot be recovered during magnification. Magnification can be performed by rasterizing the vector representation on an image grid of a higher resolution. Editing first requires an interactive selection of patches in regions of interest. This is followed by modifying TPS basis values or geometry control points in regions of interest. As shown in Fig. 3 and 3.18, both color discontinuities and smooth variations are preserved in zoomed-in images. It largely attributes to the precise feature alignment, the absence of which leads to magnification artifacts such as the color bleeding in Fig. 3.16 caused by misalignment of vector representations to image features. Without precise alignment, color values of a pixel cannot be guaranteed to be derived only from the correct side of the true image feature when an image is magnified. Editing suffers the same problem as in magnification, as shown in Fig. 3.17.

Limitations. There are certain aspects of our algorithm that can be further improved. Currently there is no scale information associated with the detected edge features. Since the scale of an edge signifies its importance, we could perform scale-space edge detection [38] and set a scale threshold to prune less important edges before initial mesh construction. We could also relate the quality of the vectorized image with the scale threshold. A higher quality vectorization is associated with a lower scale threshold to preserve more curvilinear features. Second, our thin-plate spline fitting has not been fully optimized. Currently, the center locations of the basis functions are determined by uniform subdivision of the parametric

Figure 3.16: Magnification: original resolution ×16. Left: an accurate magnification by precise feature alignment. Middle: color bleeding caused by misalignment of vector representation to the feature. Right: bicubic interpolation.

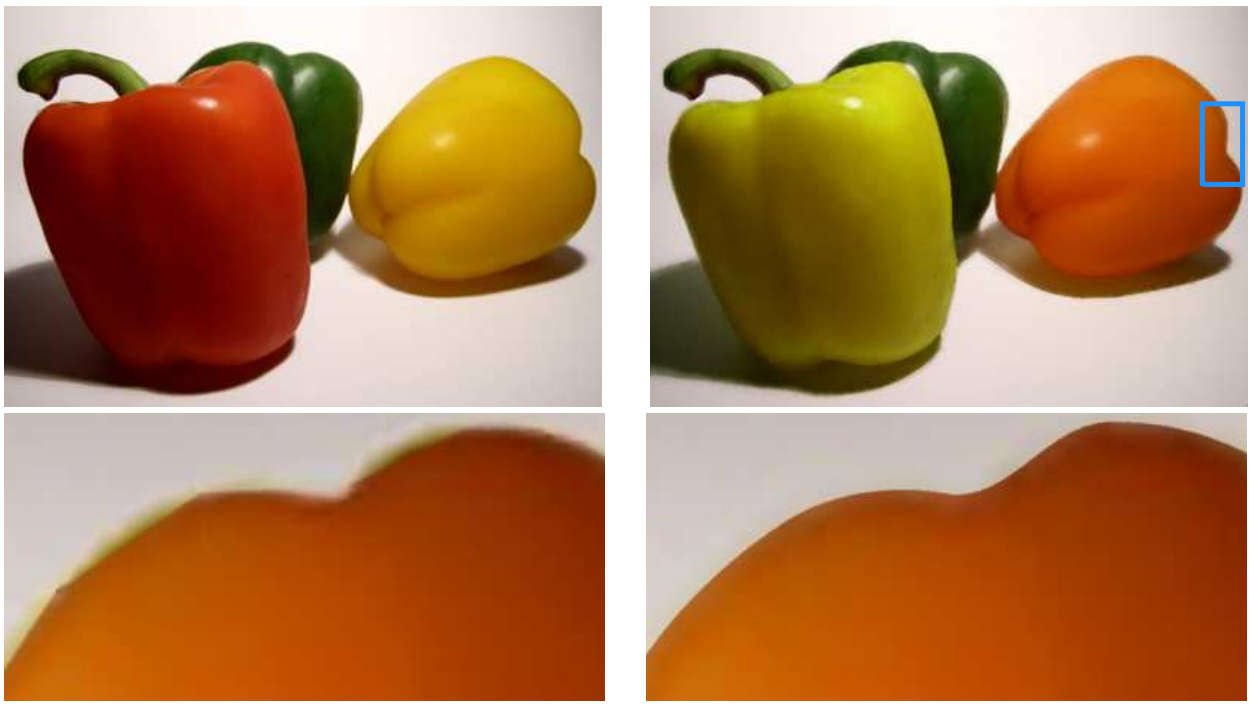


Figure 3.17: Top left: original image. Top right: color editing result. Bottom left: closeup (×4) of color editing with misalignment of vector representation to image features. Bottom right: closeup (×4) of color editing with precise feature alignment.

domain of the patch, and fixed thereafter during patch color fitting. We could formulate a nonlinear optimization that replaces the current linear solver to search for optimal center locations of the bases to further reduce fitting errors. Such nonlinear optimization would be more expensive though. The number of bases for each patch could also be adaptively determined according to the magnitude of fitting errors. Also, there is still room to accelerate our vectorization algorithm using multi-core processors.

3.7 Conclusion

In this paper, we have introduced an effective vector-based representation and its associated vectorization algorithm for full-color raster images. Our representation is based on a triangular decomposition of the image plane. The boundaries of a triangular patch in this decomposition are represented using Bézier curves and internal color variations of the patch

are approximated using thin-plate splines. Experiments and comparisons have indicated that our representation and associated vectorization algorithm can achieve a more accurate and compact vector-based representation than existing ones. A real-time GPU-based algorithm has also been developed for rendering vectorized images.

Figure 3.18: Far left and far right: original image. Mid left and mid right: reconstructed image (×1). Middle: magnification (×4).

Figure 3.19: Additional vectorization results. The first and the third are original images, followed by their reconstructed results respectively.

	# patches	mean error
Fig. 3.2(h) PEPPERS	219	0.87
Fig. 3.13 FLOWERS	380	0.98
Fig. 3.14 TEAPOT	525	0.98
Fig. 3.18 DOLL	661	1.45
Fig. 3.19 FACE	782	0.77
Fig. 3.19 GOLDFISH	925	1.61
Fig. 3 MAGNOLIA	1093	1.23
Fig. 3.18 BUDDHA	1687	1.28
Fig. 3.15 Lena	2011	2.40

Table 3.1: Statistics for image vectorization.

Chapter 4

Interactive Texture Selection for Image and Video

Chapter 3 deals with an automatic algorithm that vectorizes an raster image. In practice, however, many image processing problems cannot be solved in a completely automatic manner. Image classification and editing, for example, usually needs user interaction in order to achieve a satisfactory result. In such cases where manual input is required, how to minimize the user interaction is of significant importance. The following project proposes a framework that selects desired textures and textured objects from an image or video, with only moderate user interaction.

4.1 Introduction

Object selection and cutout from images and videos have proven to be a vital technology with many applications in computational photography, image synthesis as well as special visual effects for film making. These applications typically require pixel-level accuracy. With today's image processing and computer vision methods, computers still need human assistance in successfully performing this task at such a high level of accuracy. Hence, there has been much work on interactive object selection and cutout [39–42]. Such work typically cuts out a single connected object which is assumed to have a global color distribution model, such as a mixture of Gaussians.

In this paper, we focus on texture and textured object selection from a video. Here texture refers to a unique local spatial arrangement of colors, such as the stripes on a zebra. There are reasons why existing techniques are not suitable for texture or textured object

selection. *First*, since two different texture patterns may observe the same global color distribution, a global color distribution model would be inherently ambiguous in texture discrimination. *Second*, texture regions may be fragmented. Overly emphasizing spatial coherence to compensate the weakness in color distribution models would not work well.

Figure 4.1: Texture selection and editing examples. Regions with cloth textures are first cut out using our texture selection method, and then replaced with new textures using texture cloning.

We propose to perform interactive texture and textured object selection using a supervised classifier interactively trained using active learning. A texture classifier demands training examples interactively supplied by the user to improve its performance. Instead of the user searching for new training examples, we adopt the active learning methodology [43–45] which makes the user and the machine cooperate to find minimal data sufficient to train a good classifier. The machine plays a more active role, trying to figure out which subset of the originally unlabeled data, once labeled, would be the best training data. Once such data has been discovered, the user is asked to provide their correct labels. In this mode, the user only needs to answer a small number of queries from the machine. Hence user intervention is much reduced. We have confirmed experimentally that training data automatically chosen by the machine can indeed improve classification performance more significantly than those chosen by the user. Required by interactive performance, we adopt boosted decision trees as the classifier whose training sessions can be quickly performed in an interactive rate.

Powerful classification algorithms still require discriminative data. There has been ex-

tensive work on texture descriptors in the image processing and computer vision literature [46; 47]. Instead of a global color distribution model, we adopt local texture descriptors based on responses to an oriented filter bank. Such texture descriptors have proven to be effective in texture discrimination, and are actually used as input to our texture classifier.

The interactively trained texture classifier achieves a high success rate. In the event of minor classification errors, we rely on the graph cut algorithm to remove such errors by minimizing a revised objective function which effectively incorporates intermediate results from the classifier. More specifically, the likelihood function is formulated according to the confidence value returned by the classifier. In our experiments, such a formulation of the objective function has proven to be very effective in the refinement of the intended texture selection.

Our system remains responsive throughout the interactive session, and is not sensitive to an increasing number of frames in video. With the guaranteed scalability, the user can now select complicated texture regions and textured objects throughout the whole video by drawing only a few initial scribbles followed by a few more to label the query data. Afterwards, one can conveniently achieve color editing, compositing, and texture cloning operations on the selected regions or objects.

4.2 Related Work

Unlike video texture selection, there is a large body of literature on image texture classification and segmentation [48–53]. Since local pixel configurations provide vital information of a texture, one popular method for texture discrimination is based on filtering which can effectively characterize local patterns. Common filters used for this purpose include first and second derivatives of the Gaussian [50], Gabor filters [49], and wavelet decomposition [48]. Both classifier based [51] and boosting based [52; 53] techniques have been developed to automatically perform texture classification or segmentation. Nevertheless, these methods

have not reached our desired level of accuracy.

Many interactive techniques have been successfully developed for object cutout from still images [39; 40; 54–57]. These techniques are primarily designed for image regions with spatial coherence rather than textures with much less coherence. The Gaussian mixture model commonly used for color distribution has been extended to have discriminative power for textures in [58; 59]. In particular, a Gaussian mixture model for principal components extracted from texture description has been adopted in [58]. Nevertheless, only three leading principal components were used in [58]. In comparison, since we adopt a supervised classifier with a fast training algorithm, it has become possible for us to have a texture descriptor with more discriminative power by exploiting at least an order of magnitude more principal components.

An important inspiration of our technique came from user-defined scribbles which have been extensively exploited among aforementioned interactive techniques for images. As described in [40; 58], it is promising to have the user explicitly mark only positive or negative points while the rest of the image is implicitly labeled as the opposite. However, one-sided initial scribbles do not guarantee to converge if the color distribution of the background and foreground are somewhat similar. [57] proposed a novel segmentation technique that computes for each pixel a weighted distance to user scribbles, but the algorithm relies on a relatively large number of initial and subsequent scribbles by the user. Our method does not need a large number of scribbles or deliberate positioning of them. And both positive and negative scribbles are used in order to train a more accurate classifier.

Scribbles have also been utilized in many other topics, including colorization [60–63] and local color adjustment [64]. Texture descriptors based on Gabor filters were adopted in [62] to assist level set propagation. Texture patch similarity constraints were taken into account in [63] to solve a labeling problem. [64] trains a boosting-based classifier for edge-aware interpolation. Unlike the classification method in [61], we perform active learning directly on the input data and do not need precomputed examples from a database. Note

that colorization and segmentation are two related but different problems. While segmentation demands an accurate object boundary to be found in support of object cutout and compositing operations, colorization only alters pixel colors, and, in most cases, does not aim at a precise boundary in pixel resolution.

A certain level of user guidance has been incorporated with static and dynamic texture editing. Neighborhood-based self-similarity were exploited in [65] to quickly select features intended for further editing. But there was no supervised classification except a threshold for neighborhood similarity. User guidance has also been exploited in dynamic texture editing to alter the trajectory [66] or dynamics [67] in the original images.

Video texture classification poses a harder problem. Unfortunately, all interactive cutout methods mentioned above were precisely designed for static images. To our knowledge, no generalization from static image to video is ever clearly mentioned due to the fact that the user's workload may potentially increase in proportion to the number of video frames. Furthermore, in the relatively little literature of interactive video object cutout, [41; 42] were not originally designed for texture or textured object selection. More importantly, these techniques inevitably suffer from the scalability problem, as they treat the costly 3D graph cut as an indispensible part of their interactive session. Our method well handles this problem by freeing the whole interactive session from the graph cut overhead. A revised graph cut algorithm is only needed as a postprocessing step in our system, hence the interaction is not subject to the growing number of video frames. This is particularly useful for video texture selection with moderate user interaction.

4.3 Overview

During the preprocessing stage, spatially and temporally nearby pixels sharing similar attributes are grouped into *supervoxels*. In the special case of a static image, each supervoxel is actually a superpixel. For each supervoxel, a vector texture descriptor is obtained by

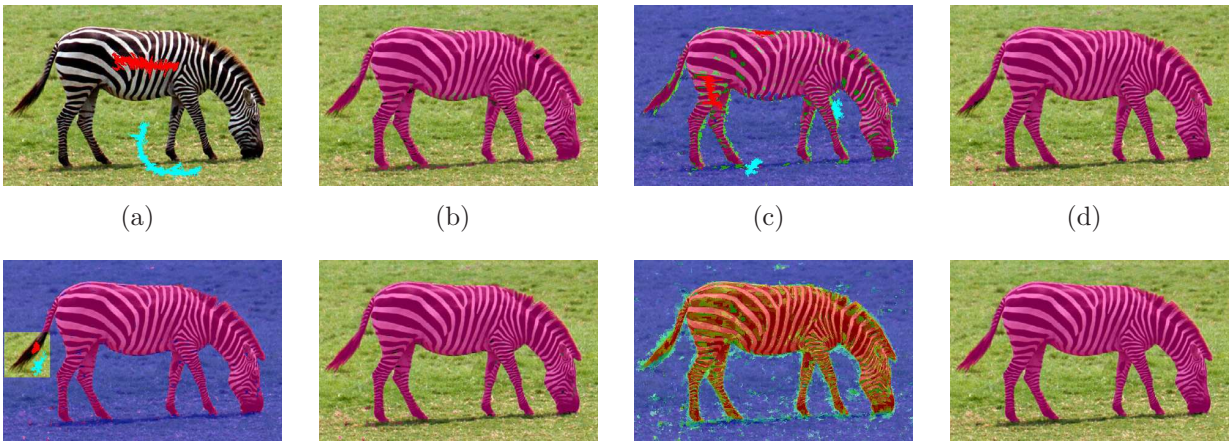


Figure 4.2: The texture selection procedure. (a) Initial scribbles. (b) Tentative classification by the first classifier. (c) Query points (green) and user provided labels (solid red and cyan). In most cases, a few scribbles are sufficient to have a tangible improvement of the classifier's performance. The user can answer the query by covering the uncertain area with positive or negative scribbles. Note that the user does not have to label all query points. (d) Improved classification after the query. (e)&(f), a local classifier is trained to refine the tail (optional). (g) Signed confidence map of the final classification. A continuous value in $[-1, 1]$ is computed for each pixel. -1 (blue) and 1 (red) indicate the highest confidence in classification, while 0 (green) implies high uncertainty. (h) Refinement by the revised graphcut algorithm. Note that our scribbles have rough and irregular boundaries because they are displayed as the union of covered supervoxels.

calculating its response to an oriented filter bank.

During the online user interaction stage, the input video is presented to the user as a sequence of frames. The user draws a few scribbles on one frame indicating the desired texture regions. The supervoxels covered by these initial scribbles serve as the set of training data on which the first tentative classifier is trained and applied to the unlabeled data. The classifier also computes a confidence value for each of the unlabeled supervoxel. It then chooses a small number of supervoxels with lowest confidence, and queries the user for their correct labels. The user's feedback is used to obtain an improved version of the classifier, which again chooses the least confident supervoxels to question the user. This iterative procedure normally runs for a few times before the classifier performs sufficiently well on the video.

Usually, the improved classifier is not completely accurate in prediction. It is likely to have minor misclassifications. In the video, these labeling mistakes are observed as noisy

pixels inside and outside the intended region, for which further refinement is needed. We refine texture selection results by breaking supervoxels into pixels and applying a revised graph cut algorithm with an innovative weighting scheme based on the confidence values obtained by our classifier. For videos, graph cut is applied to all frames as an offline batch postprocessing step once the online interactive training session for the classifier is over. For static images, we include graph cut as an additional step in the interactive training session since graph cut has reached interactive performance on single images. We alternate classifier training and graph cut on the input image until convergence. The whole procedure is demonstrated in Figure 4.2.

Though mainly designed for video texture selection, our system is a perfect fit for image cutout too. For convenience of presentation, most illustrations shown in the paper are experiments on images. Please refer to supplemental clips for video selection results.

4.4 Preprocessing

To make run-time texture selection more efficient and responsive, we preprocess the video using the algorithm in [68] so that supervoxels are formed as working units for following operations. A supervoxel corresponds to a small group of geometrically close pixels that share similar color and intensity attributes. Instead of processing every single pixel within the texture video, we only process a representative pixel within each supervoxel, and let the rest share the same result. For similar reasons, superpixels have been found useful in [39; 69]. As in [41; 42], we generalized superpixels to supervoxels so nearby pixels across consecutive frames are also incorporated to enhance the temporal coherence of the video. We set an upper bound on the number of pixels within a supervoxel (typically 50), and confine each supervoxel within a cube to prevent long slivers and spiral-like shapes.

The second step of preprocessing is to obtain texture descriptors for each supervoxel. Oriented filter banks [46; 47] have proven to be an effective tool to characterize textures. Our

learning algorithm primarily uses local statistics of oriented filter responses to differentiate textures. We apply 24 Gabor filters [46] at 6 orientations and 4 scales at every pixel and such filtering is performed for each of the three color channels separately. Thus, every pixel has a 72-component filter response vector after filtering. We compute the mean and standard deviation of each channel over all pixels within every supervoxel to obtain a 144-component vector. Note that the Gabor filters can gather texture information not only within a supervoxel but also in areas surrounding the supervoxel because the support region of the filters extends well beyond the supervoxel.

At the representative pixel of every supervoxel, a color histogram is also extracted from its 7×7 neighborhood. We used 10 bins for each color channel, and the 30-component histogram vector is then concatenated with the 144-component vector. It is reduced to a shorter vector by principal component analysis. This shortened vector, in juxtaposition with the mean and standard deviation of each color channel, forms the texture descriptor for every supervoxel. The inclusion of color information in descriptors effectively alleviates the tendency of blurring the actual boundary when only texture features are used. In our experiments, we have used different descriptor lengths for different examples, normally ranging from 30 to 60. These descriptors are the data points fed to Algorithm 1 during interactive training.

As an option, we have also experimented with the oriented filter bank in [47], another filter bank known for its success in texture discrimination, and achieved comparable results.

4.5 Active Learning Based Selection

In this section, let us first review the active learning approach. Suppose the input data to supervised classification consist of a set of tuples, $(\mathbf{x}_1, l_1), (\mathbf{x}_2, l_2), ..., (\mathbf{x}_n, l_n)$, where $\mathbf{x}_i \in \mathbf{X}$ denotes one of the data, and l_i denotes the class label of that datum. $\mathbf{x}_i$ is typically a vector in a multi-dimensional space, $\Re^d$. If one is only concerned with two classes of data, $l_i \in L = \{-1, +1\}$. Classification is then equivalent to finding a boundary surface in this

Algorithm 1: The Query-by-Boosting algorithm.

Data: $\mathbf{X}$ (unlabeled point set), N (number of trials) and m (number of query candidates)

begin

  Initialize $S_1 = \{(\mathbf{x_1}, l_1)\}$ for random $\mathbf{x}_1$;

  for *i=1 to N* **do**

    Obtain a boosted classifier H_i on S_i;

    Randomly generate a set of m points, $C \subset X \backslash S_i$;

    Pick a point $\mathbf{x}^* \in C$ with the minimum margin:

      $\mathbf{x}^* = \arg\min_{\mathbf{x} \in C} |\sum_t \alpha_t h_t(\mathbf{x})|$;

    Query the label at $\mathbf{x}^*$, $l(\mathbf{x}^*)$;

    Set S_{i+1}=append$(S_i, (\mathbf{x}^*, l(\mathbf{x}^*)))$.

  end

end

Result: the boosted classifier at the last iteration, H_N.

d-dimensional space so that data with different labels do not lie on the same side. As we can imagine, data points close to the boundary play a much more important role in defining the shape of the boundary than those further away. This is especially true when data with different labels interlace, which gives rise to a jagged decision boundary. A precise definition of the boundary requires a much denser sampling of the space around the boundary than anywhere else.

Active learning [43–45] is exactly motivated by this observation. It suggests a way for the learning algorithm to proactively choose training data instead of passively receiving them. Once an initial boundary is defined, it keeps refining the boundary automatically by sampling more and more points around it. Obviously, labels for these additional data points need to be supplied by the user. But how can the learning algorithm know which points lie close to the boundary? A typical strategy is to train multiple classifiers and conduct an internal voting to determine the uncertainty of a data point. A point is considered close to the decision boundary when the voting result becomes close to 50-50, which indicates a high uncertainty.

Active learning needs to work with a fast classification algorithm to achieve its goal. In

Algorithm 2: The AdaBoost algorithm.

Data: $(\mathbf{x}_1, l_1), ..., (\mathbf{x}_n, l_n)$ where $\mathbf{x}_i \in \mathbf{X}$, $l_i \in \{-1, 1\}$

begin

Initialize distribution, $D_1(i) = 1/m, i = 1, ..., n$;

for *t=1 to T* **do**

Using D_t, train base classifier $h_t : \mathbf{X} \to \{-1, 1\}$
with small error $\epsilon_t = \Pr_{D_t}[h_t(x_i) \neq l_i]$ on D_t;

Set $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t l_i h_t(\mathbf{x}_i))}{Z_t}$,
where Z_t is a normalization factor,
and $\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$.

end

end

Result: the final classifier, $H(\mathbf{x}) = \text{sign}\left(\sum_t \alpha_t h_t(\mathbf{x})\right)$.

this paper, we adopt decision tree classifiers enhanced with the AdaBoost algorithm [70] (Algorithm 2). Before binary classification, the classifier first computes a margin, $|\sum_t \alpha_t h_t(p)|$, which is also defined as the *confidence*. Boosted decision trees can be trained interactively while achieving a strong discriminative power. Note that other classifiers with a fast training phase can also be adopted, such as the soft-margin linear SVM. A method that integrates boosting with active learning has been developed in [44]. An outline of the method is shown in Algorithm 1. It assumes the knowledge of the entire pool of unlabeled data. Since this pool may be very large, during each iteration, it generates only a random subset of the unlabeled data and the uncertainty of each point in the subset is indicated by the margin of the weighted votes cast by all the base classifiers trained by AdaBoost. The method finally chooses the point with the highest uncertainty, i.e. minimum confidence, and requests the user to provide its label. The chosen point along with its label is then appended to the set of training data.

4.5.1 Initial Feature Selection

Run-time texture selection is cast as a binary supervised classification based on Algorithm 1 with certain revisions. In the current context, the set of unlabeled data, $\mathbf{X}$, consist

of all supervoxels within the input video. The machine randomly selects an initial video frame to start with and let the user draw one or more scribbles within that frame to mark desired features. Texture descriptors at supervoxels touched by these scribbles (note that a supervoxel might straddle across several frames) are marked as positive training data. The user also draws one or more additional scribbles to mark negative training data. Positive and negative labels are conveniently communicated using left and right mouse buttons, respectively. We apply the K-means algorithm to cluster supervoxels covered by the positive and negative scribbles respectively, and uniformly sample each cluster. Given both types of sampled training data, an initial boosted classifier can be obtained as in Algorithm 2. Unlike Algorithm 1, we allow multiple initial training data. Note that initial scribbles do not increase with respect to the number of video frames, as it is confined to one frame and requires no more scribbles than a single image does.

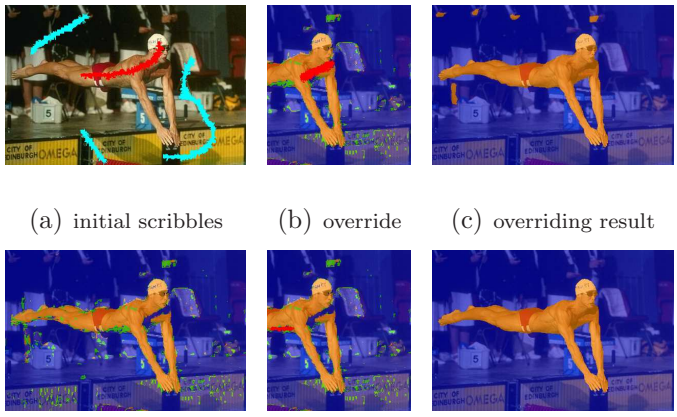
4.5.2 Automatic Query

Once an initial classifier has been obtained, as in Algorithm 1, we run multiple subsequent iterations with user query to refine the initial classifier. Each iteration proceeds as follows. Instead of generating a random subset of the unlabeled supervoxels from the entire video as in Algorithm 1, we only generate query candidates from one randomly chosen video frame. All user interactions within that iteration will be focused on the data from that frame only because shuffling among multiple frames would be too distracting. If the number of supervoxels within that frame is sufficiently small, all of them become query candidates; otherwise, a random subset of these supervoxels are generated as query candidates. The classifier from the previous iteration is applied to generate confidence values for all query candidates.

This strategy has important advantages. The entire video consists of a large number of supervoxels, and a random subset of the entire video may fail to include important texture descriptors that lie close to the decision boundary, making active learning less effective. On

the other hand, supervoxels from one single video frame provides a much narrowed scope. Sampling within the much smaller pool of supervoxels significantly increases our chance to choose points close to the decision boundary as query points.

Our algorithm offers a high degree of flexibility to condition the training of classifiers. Instead of choosing only one query point at a time as in Algorithm 1, we allow multiple query points every iteration to speed up the convergence. We choose from the candidate pool all points with a margin smaller than a prescribed threshold (∼0.1-0.15) to inquire the user. Supervoxels covered by these query points are all shown in a special color. Instead of providing the correct label to one query point at a time, the user can draw relatively long scribbles to paint multiple query points with the same label. Painting with labels can be careless because only query points are affected by the scribbles and the rest of the supervoxels are masked.



(a) initial scribbles (b) override (c) overriding result

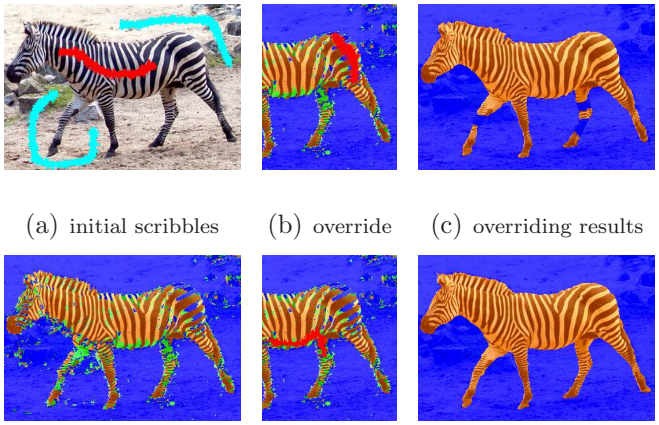
(d) tentative classes (e) query (f) result by query

Figure 4.3: Initial scribbles shown in (a) are used to train a tentative classifier, whose classification result is shown in (d). Displayed in green is the automatic query by the machine. (b) illustrates the overriding scribble (in red) based on the classification in (d). It manages to classify the swimmer, but also introduces misclassifications in the background as a side effect shown in (c). (e) shows the user's feedback (in red) based on the query in (d). This query significantly improves the accuracy of the classifier without negatively affecting previously correct labels. (c) and (f) are final results after applying graph cut.

In addition, the user can directly draw overriding scribbles over incorrectly labeled data in addition to labeling the query data. The input of such overriding scribbles corrects partial classification errors in previous rounds, if any, and leads to a more accurate classification in the next round. Nevertheless, overriding scribbles are applied rather infrequently. In practice, the percentage of such scribbles is typically around 10%.

Once the learning algorithm has obtained the labels of the query points and possibly overriding scribbles, an improved classifier is trained using the newly labeled supervoxels in the current iteration together with a uniformly sampled subset of the labeled supervoxels from previous iterations, and the procedure is repeated.

4.5.3 Local Classifiers



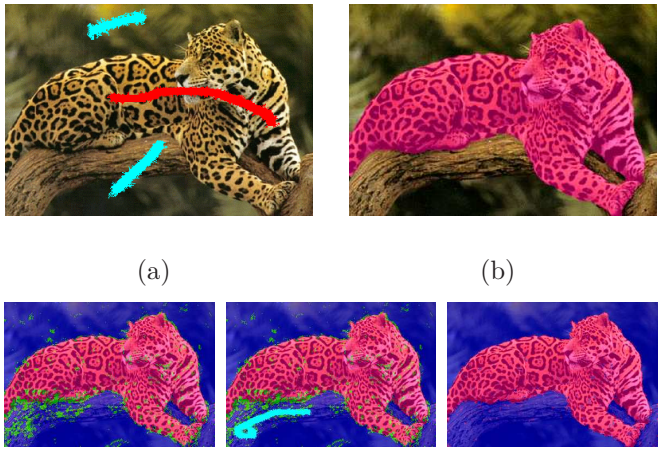
(a) initial scribbles (b) override (c) overriding results

(d) tentative classes (e) query (f) result by query

Figure 4.4: In (b), the overriding scribble (in red) on the zebra's rear part have well corrected nearby misclassifications. But it eventually leads to a classifier that fails to recognize similar stripe patterns on the zebra's legs in (c). (e): In contrast, by feeding correct labels (in red) to query points (in green), we manage to build a stable classifier with high accuracy. (c) and (f) are the final results after applying graph cut.

A video sometimes has a region with abundant texture details that requires considerably more training data than other regions do. When this is the case, our algorithm is inefficient because the candidate query points are randomly (and thus uniformly) generated and the region in question tends to be undersampled. We handle the situation by providing the user with an option to define a 3D rectangular region of interest within the video, and train a local classifier for the specified region. The local classifier works exactly the same way as the global one does, only confined within the user specified region. Apparently, the detailed region is much smaller compared to the whole video, resulting in a faster rate of convergence. It is therefore a good practice to refine regions of detailed textures with a local classifier after the classification for the entire video.

4.5.4 Validation of Active Queries



(a) (b)

(c) (d) (e)

Figure 4.5: (a) illustrates initial scribbles used for training the tentative classifier. Its classification result is shown in (c), where regions in green are displayed as an automatic query by the machine. As we see in (d), user-supplied labels (in cyan) are only a subset of all query points, avoiding locations not obvious to the eye. An improved classifier is then trained, and the new classification is shown in (e). (b) is the final result after applying the graph cut to the classification in (e).

In this section, we demonstrate that active queries at data points close to the decision boundary can most effectively and quickly improves the accuracy of the classifier. As a result, by providing the queries automatically as well as accelerating the convergence of the classifier, active learning can tangibly reduce the amount of user interaction.

We compared the effectiveness of active queries and overriding scribbles in improving the overall classification accuracy. In our experiments, we have observed that active queries in a local image region not only improve the classification results in that local region, but also improve the decision boundary on a more global scale. For example, in Figure 4.3 and 4.4, the labels provided to the active query have succeeded in correcting all misclassifications. Meanwhile, the improved classification results remain stable meaning that active queries in one region tend not to negatively affect previously correct labels in other regions. On the other hand, overriding scribbles sometimes tend to overemphasize textures they cover, and result in undesirable side effects elsewhere, as shown in Figure 4.3. In addition, as shown in Figure 4.4, overriding scribbles also exhibit an inability to simultaneously correct the erroneous labels of similar patterns that may not be spatially close.

Because of the ability of active queries in improving classification accuracy on a global scale, it is not necessary for the user to provide labels at all query points. In fact, tangible improvements within an entire image can be achieved with a very small number of queries. Thus, the user can choose not to label the query points at ambiguous locations, such as those close to the boundary of the intended texture region. As shown in Figure 4.5, user-supplied labels at exterior query points actually improve the classification result at the texture boundary.

4.6 Selection Refinement

As has been described in previous sections, a classifier makes the most uncertain decisions on its decision boundary. Another source of misclassification is the presence of outliers. As each data point (supervoxel) is classified according to its own texture descriptor, the relationship among neighboring supervoxels is not taken into account by the classifier. The neglect of the spatial coherence across supervoxels gives rise to occasional outliers in the video. These observations have motivated us to use the graphcut algorithm as a selection refinement mechanism at the pixel level. In addition, a refinement at the pixel level is able to break misclassified super-voxels that straddle across weak boundaries.

4.6.1 Graph Cut

Graph cut [71] has been used as a powerful image segmentation tool in recent years. The underlying idea is to treat image segmentation as a binary labeling problem. Specifically, the image induces a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. A node corresponds to a pixel in the image. There is an edge between every pair of neighboring pixels, whose weight (cost) encodes the difference of the two pixels. The more they differ, the smaller the cost. Every pixel also has links to two virtual nodes, the source representing the foreground and the sink representing the background. The weight on each link indicates how

likely it is for the pixel to belong to the foreground and background. The labeling problem is to assign a unique label l_i for each node $i \in \mathcal{V}$, i.e. $l_i \in \{\text{foreground}(=1), \text{background}(=0)\}$. The solution $L = \{l_i\}$ can be obtained by minimizing the following objective function:

$$E(L) = \sum_{i \in \mathcal{V}} R(l_i) + \lambda \sum_{(i,j) \in \mathcal{E}} B(l_i, l_j), \tag{4.1}$$

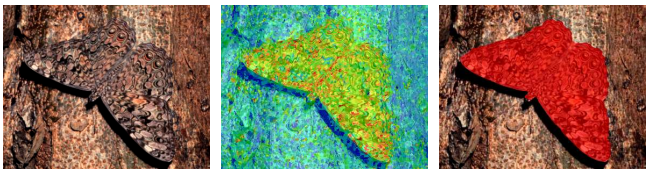
where $R(l_i)$ is a likelihood energy, encoding the cost when the label of node i is l_i, and $B(l_i, l_j)$ is a prior energy for boundary penalty, denoting the cost when the labels of adjacent nodes i and j are l_i and l_j respectively. If we set the weight of edge (i, j) to $B(l_i, l_j)$, the weight of the link between node i and the source to $R(l_i = 1)$ and the weight of the link between node i and the sink to $R(l_i = 0)$, finding an optimal solution of (5.1) is equivalent to finding a cut with minimal cost in the graph. More details can be found in [71] and [40].

We would like to make use of the interactively trained texture classifier in the likelihood energy of the graph cut algorithm. Since the result returned by a classifier is essentially a complicated nonlinear function of the training examples, it can be considered as an accurate example-based parametric model built upon texture descriptors. Therefore, instead of using a global color distribution to set the source/sink link cost as in [40], we opted for the signed confidence value, $\sum_t \alpha_t h_t(p)$, computed in Algorithm 2, to relate to the likelihood energy. Specifically, we set $R(l_i = 0) = \max(0, -\ln \max(0, 0.5 + 0.5 \sum_t \alpha_t h_t(p)))$ and $R(l_i = 1) = \max(0, -\ln \max(0, 0.5 - 0.5 \sum_t \alpha_t h_t(p)))$.

To maintain spatial coherence and remove classification outliers, we adopt the conventional scheme of assigning edge cost where the color difference between adjacent pixels is used, i.e. $B(l_i, l_j) = |l_i - l_j| \exp\left(-\frac{\|\mathbf{c}(i) - \mathbf{c}(j)\|^2}{2\sigma_c^2}\right)$ where $\mathbf{c}(i)$ represents the color vector at node i. Figure 4.6 illustrates the result using the graph cut method.

4.6.2 Graph Cut Integration

Considering the difference in computational cost when the graph cut algorithm is respectively applied to a single image and a video, it is integrated with the rest of the texture selection process in two different schemes.



(a) Original Image (b) Confidence Map (c) Graphcut

Figure 4.6: With difficult examples where foreground and background color distributions are similar, the interactively trained classifier results in misclassifications at random locations as in (b), where each pixel is assigned a signed confidence value. (c) Graphcut based on signed confidence values computed by the classifier proves to be a powerful refinement method.

Since the graph cut algorithm can achieve interactive performance on a single image, it is interleaved with classification and active learning to iteratively refine texture selection results. More specifically, starting from a tentative classification C_i we perform active learning and automatic queries to obtain an improved classification C_{i+1}. A graph based on C_{i+1} is then formulated as in section 4.6.1, followed by a minimum cut. Note that the graph is defined at the pixel level and all pixels within the same supervoxel receive the same value related to the likelihood energy. The foreground and background regions from the minimum cut are then uniformly sampled. These sample pixels together with pixels on user scribbles and query feedbacks are collectively used as positive and negative training data, from which a new classifier is trained (Algorithm 2). In the event a supervoxel is broken into foreground and background regions by the graph cut, majority voting is performed on the supervoxel to determine its new label. This procedure is repeated until convergence. Note that for images with relatively simple textures, we can skip the active query step within each iteration and simply alternate between classifier training and graph cut to achieve satisfactory results (Figure 4.8).

However, a graph cut for dozens or even hundreds of video frames is too computationally expensive to achieve interactive update rates. Therefore, we take the graph cut algorithm out of the interactive session and simply apply it as an offline postprocessing step in video

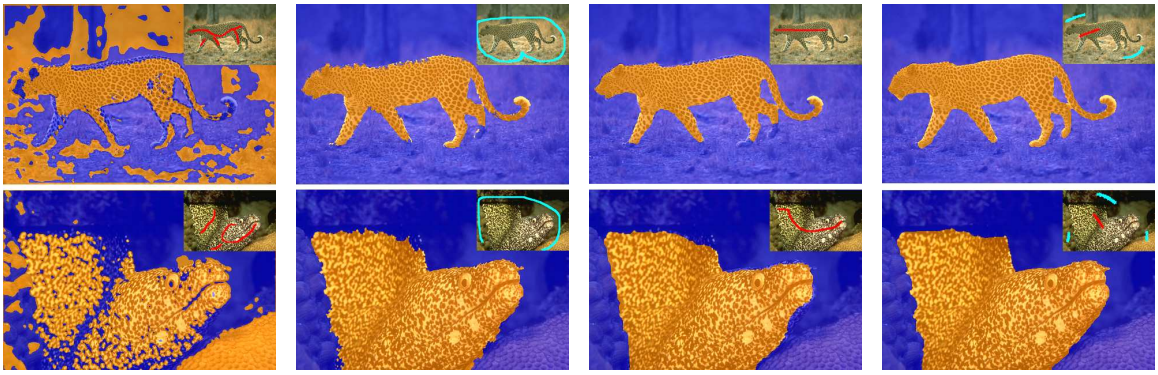
Video (or Image)	Size	Feat. Dim	Prep	Training		Graphcut	User Interaction
				total	per query		
flower	$352 \times 288 \times 30$	30	15 min	5.1 sec	0.26 sec	12 sec	6 min
sea plant	$328 \times 264 \times 104$	30	54 min	10.8 sec	0.31 sec	1 min	23 min
smoke	$455 \times 355 \times 33$	30	19 min	3.7 sec	0.37 sec	8 sec	5 min
bubbles	$328 \times 264 \times 32$	45	17 min	6.4 sec	0.34 sec	17 sec	10 min
jaguar	$352 \times 288 \times 35$	60	21 min	7.8 sec	0.52 sec	21 sec	13 min
zebras	$321 \times 252 \times 107$	50	65 min	16.5 sec	0.41 sec	1.1 min	30 min
curtain	$396 \times 271 \times 30$	60	22 min	6.9 sec	0.34 sec	18 sec	8 min
shirt	$312 \times 222 \times 35$	60	25 min	7.9 sec	0.37 sec	21 sec	11 min
monarch butterflies	529×347	60	69 sec	3.5 sec	0.37 sec	3.8 sec	67 sec
fabrics	442×349	60	59 sec	3.2 sec	0.31 sec	4.2 sec	50 sec
dress	326×548	40	52 sec	2.2 sec	0.29 sec	3.0 sec	48 sec
big cat	371×350	40	63 sec	3.1 sec	0.43 sec	3.4 sec	78 sec
group zebras	457×297	55	55 sec	2.7 sec	0.38 sec	2.2 sec	96 sec
fungi	538×339	40	64 sec	0.85 sec	0.21 sec	-	27 sec

Table 4.1: Timings for video sequences and static images. Performance measurements were taken on a 3.8GHz Pentium 4 processor. "Feat. Dim" denotes the dimensionality of texture descriptors; "Prep" denotes the preprocessing time; "Training" denotes the training time for classifiers. It has two subcolumns. 'total' means the accumulated training time for a dataset while 'per query' means the average training time for a new classifier after each round of query. "User Interaction" denotes the total artist time. Graph cut is performed as offline postprocessing for videos. However, for static images summarized in the bottom half of the table, time spent on 'Graphcut' indicates the accumulated time from multiple iterations.

texture selection. More specifically, once multiple iterations of active learning have been performed to obtain a satisfactory classifier and its classification results, the user runs a postprocessing step where a framewise graph cut is applied at the pixel level to further refine selected texture regions. Temporal coherence across consecutive frames is secured by the supervoxel mechanism (Section 4.4) in the classification stage, and will not be adversely affected by the framewise graph cut.

4.7 Experimental Results

We have experimented with a series of video sequences as well as static images on a 3.8GHz Pentium 4 processor. As described in Section 4.5, our classifier only requires shallow decision trees of moderate accuracy, whose performance are later boosted by Adaboost and active learning. Building such decision trees (usually 3 or 4 levels in depth) using training data with reduced dimensionality is very fast. This gives rise to the computational efficiency shown in the Training column of Table 4.1, and it normally takes 0.2 to 0.6 seconds to train



(a) Gabor filtering (b) GrabCut (c) Principal Channels (d) Our method

Figure 4.7: Comparison among multiple object selection methods (selection based on similarity of Gabor filter responses, grabcut [40], p-channel [58], and our lazy texture selection). the smaller images in the upper right corner show initial scribbles needed for each method. Our method (the last column) excels in precisely identifying object boundaries (e.g. paws of the leopard and the mouth of the fish).

a completely new classifier for video sequences as well as for static images after each round of interactive query and labeling. Because the training time for each classifier is well below one second, labeling and training stages have thus been seamlessly integrated and have reached an overall interactive rate. Because the total amount of training data collected from initial labeling and subsequent query sessions does not increase significantly for videos, the time needed for each round of training does not have obvious difference between videos and static images.

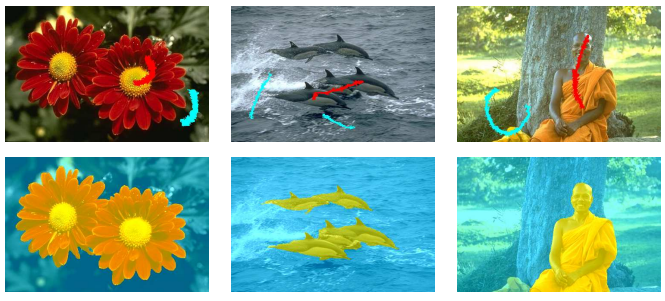
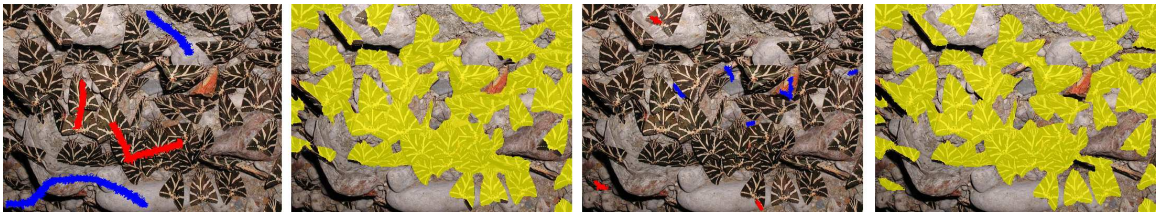


Figure 4.8: With initial scribbles illustrated in the first row, our iterative method can successfully select desired regions without any further user interaction.

One major advantage of our video texture selection method over existing video object selection methods [41; 42] is that ours keeps graph cut out of the interactive session and only applies it as an offline postprocessing step. Existing methods integrate graph cut as an indispensable part of the interactive session, making a prompt response vulnerable to the growing size of the video data. In contrast, each iteration of our interactive session only

(a) Selection using GrabCut. Obvious misclassifications are indicated in the last image.



(b) Results of our lazy texture selection method.

Figure 4.9: Comparison of interactive texture selection methods.

includes fast training and classification, and exhibits very good response time and scalability, which are crucial for video editing. It compares favorably with the performance of each iteration of existing video object selection methods whose performance deteriorates significantly with an increasing number of video frames due to the graph cut within the interactive loop. If we discount computation time within each iteration, the total user interaction time of our texture selection method is comparable to that reported in [41; 42] which were not originally designed for texture selection.

Our algorithm has two essential components, namely, active learning (query) and a supervised classifier on which active learning is based. We first verify the performance of the supervised classifier. To leave active learning out of the pipeline, we adopt the iterative approach mentioned in section 4.6.2, alternating classification with graph cut without further user interaction (query) until convergence. As shown in Figure 4.8, our classifier is capable of identifying relatively simple texture patterns without active queries. To further demonstrate the discriminative power of our texture classifier, we compare it with state-of-the-art object selection algorithms. Both GrabCut [40] and P-Channel [58] use the same iterative framework while relying on the Gaussian mixture model for texture discrimination. As shown

in Figure 4.7, our classifier produces better results than other methods, especially along object boundaries. Texture discrimination based on similarity of Gabor filter responses fail to precisely recognize distinct texture patterns due to its unsupervised nature.

With a powerful base classifier ensured, we then examine if active learning and query further boost the performance on examples with sophisticated textures and scene composition. We conducted a number of comparisons with GrabCut [40] which is also allowed to accept additional user scribbles inbetween iterations. Using an equivalent number of scribbles, our active learning based method performs at least as well as GrabCut in all experiments, and consistently produces better results on examples with scattered texture regions. As shown in Figure 4.7, GrabCut does poorly in selecting such scattered texture regions because of its specific graph cut formulation based on the Gaussian mixture model. Notice that GrabCut needs considerably more user scribbles in the presence of discrete objects, and it often mistakes texture variations in the background for the object boundary. Since the video object selection methods in [41; 42] are based on the same graph model adopted in GrabCut, they exhibit the same type of problems with scattered texture regions. Our method handles these situations well, as demonstrated in this comparison as well as other examples throughout the paper and supplemental materials.

4.8 Applications

An accurate texture classification makes it possible to do further editing in user selected regions. Figure 4.11 is the result of *color transfer*, a simple technique that changes the color of the selected texture. The user picks a few seeding points within the textured object and our system clusters the rest of the pixels by finding the shortest L_2 distance to any of the seeding points in a color space. In a subsequent step, a color is chosen as the transfer target for each cluster, and the seeding point is taken as the reference. The difference between the target and the reference will be the transfer vector added to every pixel in the

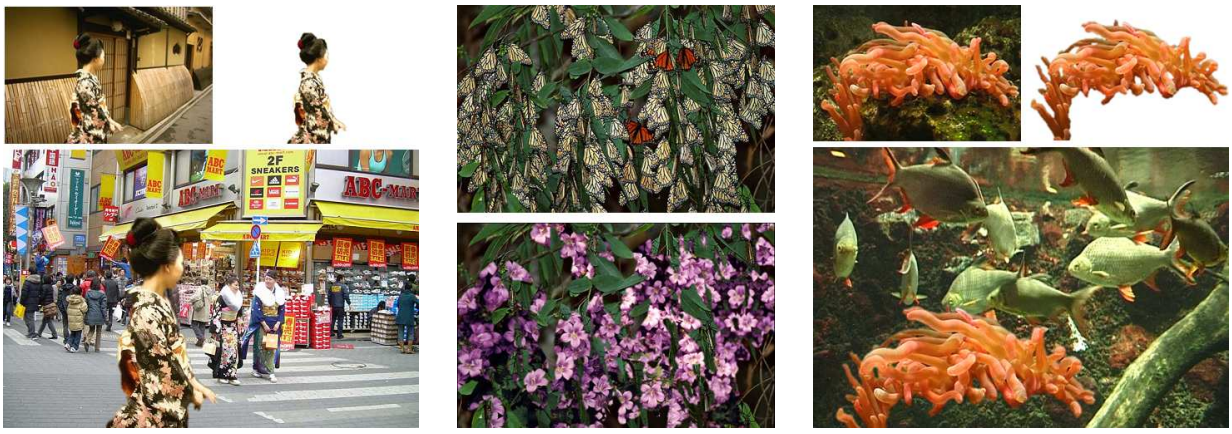


Figure 4.10: Compositing examples. Texture regions are first extracted as the foreground or background layer, the foreground layer is then composed with a different background using an estimated alpha channel within a transition region.

cluster. The transfer can be done in any color space specified by the user. In addition, we implemented a weighting scheme for pixels residing on cluster boundaries, *i.e.*interpolating colors of neighboring clusters, giving the transferred texture a softer and more natural look.

Compositing operations, for example, can be conveniently applied as shown in Figure 4.10. To achieve this, a tri-map is first constructed from the texture selection result by expanding a transition region from the texture boundary towards the interior of both foreground and background regions, followed by applying an algorithm, such as the one in [72], to estimate the alpha channel of the transition region.

Figure 4.11: Color transfer examples in user selected regions.

Texture cloning is another interesting application, in which we compute a neighborhood-based similarity between a seeding pixel and every other pixel within the selected region followed by a texture cloning procedure introduced in [65]. The idea is to

blend a new texture with the original one according to opacity values obtained from the neighborhood-based similarity computation (Figure 4.1).

4.9 Discussions and Conclusions

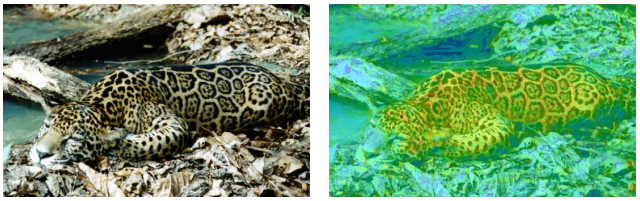


Figure 4.12: Failure example. The greenish color in the signed confidence map on the right indicates that the classifier is uncertain about the classification of most of the pixels.

In the making of our system, a few questions are open to discussion. For now, our system is yet to handle more intricate and ambiguous texture patterns such as the one shown in Figure 4.12. Hence finding a more powerful classifier that works with a broader range of textures will be an interesting topic in future study. In our compositing experiments, existing matting techniques could not give satisfactory alpha estimates when applied to boundary areas with complicated textures such as zebra stripes. In future, we hope to develop and incorporate into our system a matting procedure that better handles textured boundary areas.

In conclusion, we have described a novel interactive approach to the problem of texture selection and editing across both spatial and temporal domains. By using a robust classifier in a supervised fashion, our algorithm is able to quickly distinguish the user specified textures or textured objects. Meanwhile, it adopts the idea of active learning and puts the user in a more laid back position where only modest marking and labeling work is required. With its achievements in both solution accuracy and efficiency, our system is a convenient tool for texture selection and subsequent editing.

Chapter 5

Application: Automatic Detection for Malignant Gland Units in Microscopic Images

The classification technique introduced in Chapter 4 has wide applications in the field of medical imaging. In this project we develop effective image segmentation and classification methods for automatic detection of malignant gland units in microscopic images based on a variant of the techniques introduced in Chapter 4. Both segmentation and classification methods rely on carefully designed feature descriptors, including color histograms and texton co-occurrence tables.

5.1 Introduction

Prostate cancer is the second most frequent cause of cancer deaths among men in the US. Almost one-third of American men over 50 years old will be diagnosed with prostate cancer during their life times. Nevertheless, if diagnosed in a sufficiently early stage, prostate cancer patients have a high probability to survive. Several screening methods exist, the most reliable one being biopsy. Sample tissues from a biopsy are usually stained by Hematoxylin and Eosin (H&E) and sliced into cross sections for examination under a microscope.

Diagnosis of prostate cancer is performed by examining the glandular architecture in histological images of the specimen. Normal prostate tissue, illustrated in Fig. 5.1(a), consists of gland units surrounded by fibromuscular tissue, called stroma. Each gland unit consists of layers of epithelial cells located around an empty tubular region, named the lumen. When cancer occurs, epithelial cells replicate in an uncontrolled way to fill up the lumens as depicted in Fig. 5.1(b). In a more serious state, even the stroma disappears, giving way to replicated

epithelial cells. Therefore, the condition of gland units is the most important indicator of prostate cancer.

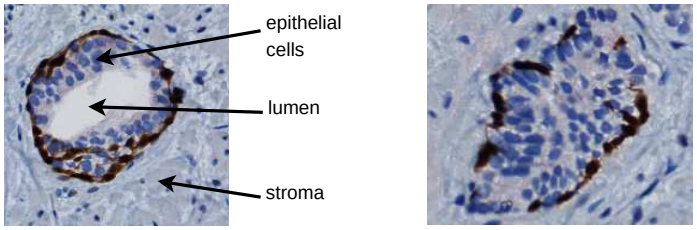
Cancerous development in gland units is usually spatially varying. Some regions may progress faster than others. In an early stage of prostate cancer, only relatively few gland units in a large region become malignant. Discovering such sparse malignant gland units in cross sections of the specimen under a microscope is a labor-intensive and error-prone task for pathologists. It would be very easy for them to miss sparse malignant units in a sea of benign ones. Nevertheless, the screening of malignant gland units is crucial for early diagnosis of prostate cancer. In this paper, we propose a method to automatically detect malignant gland units in cross-sectional microscopic images.

Achieving this task is challenging for the following reasons. First, an accurate segmentation of gland units, especially malignant gland units, is required in order to correctly collect features from them for classification. However, a gland unit is highly inhomogeneous, consisting of epithelial cells, some background tissue and the lumen. In addition, it may have an irregular shape and may not have a clear boundary. Second, criteria for classifying a gland unit should be partially based on the spatial distribution of epithelial cells within the unit. But it is not always possible to segment individual epithelial cells especially when the microscopic image does not have a sufficient resolution.

Our solution for malignant gland unit detection makes use of two cascaded classifiers. The first classifier performs pixel-level classification. It is trained to discern pixels inside gland units from those outside. The classification result provides a noisy initial segmentation of gland units. A refinement to the initial segmentation by improving spatial coherence is performed through a graph cut algorithm. Features within every segmented gland unit are then extracted and fed to the second classifier, which determines whether a gland unit is malignant or not. An important class of features we use are based on textons [47; 73], whose spatial distribution serves as a representation of the cellular structures within a gland unit. There exist two separate training stages for the two classifiers both of which share the same

set of training images. The first classifier is trained simply using a set of labeled pixels in the training images. However, the training data for the second classifier needs to be prepared using the first classifier, which segments out individual gland regions in the training images. A human expert then labels the segmented gland regions. Experimental results confirm the effectiveness of our solution.

5.2 Related Work



(a) normal gland unit

(b) malignant gland unit

Figure 5.1: Examples of gland units in cross-sectional microscopic images.

There exists much related work on prostatic image classification and cancer grading. Pitts *et al.* [74] investigated the application of gray level co-occurrence matrix techniques for the interpretation of prostate cancer lesions and identifying corresponding features on the color images of the section. Jafari-Khouzani and Soltanian-Zadeh [75] used features based on multiwavelets combined with a k-nearest neighbor classifier to classify each image into grades 2 through 5. The classification distance metric is optimized using simulated annealing. Doyle *et al.* [76] introduced a multiresolution scheme where pixel-wise Bayesian classification is performed at each image scale to obtain corresponding likelihood values. Starting at the lowest scale, they apply the AdaBoost algorithm to analyze pixels with a high probability of malignancy at subsequent higher scales. Tabesh *et al.* [77] proposed an automatic two-stage system for prostate cancer diagnosis and Gleason grading. The color, morphometric, and texture features are extracted from prostate tissue images. Linear and quadratic Gaussian classifiers were then used to classify images into tumor/nontumor classes, and further into low/high grades for cancer images. Most recently,

Huang and Lee [78] proposed two feature extraction methods based on fractal dimension to analyze variations of intensity and texture complexity in regions of interest. Each image can be classified into an appropriate grade by using Bayesian, k-NN, and support vector machine (SVM) classifiers, respectively. A review of computer technologies in detection and staging of prostate carcinoma can be found in [79].

Most of the aforementioned work performs image classification and grading without explicitly segmenting out gland units. Therefore, such methods tend to produce incorrect results when sparse malignant gland units are distributed among a large number of benign ones. In our method, we perform high-quality gland unit segmentation using state-of-the-art techniques, such as graph cut. In addition, the classification of gland units is based on local texture pattern analysis using textons, which have proven to be useful for texture description.

5.3 Gland Unit Segmentation

Our gland unit segmentation is composed of two stages, an initial binary classification followed by a graphcut based refinement. We perform texture classification on microscopic images because the micro structures of tissues and cells resemble a texture image with repetitive patterns. Since texture features for classification need to be defined over local spatial neighborhoods, we first group nearby pixels sharing similar color and intensity attributes into small patches called superpixels using the algorithm in [39].

Oriented filter banks have proven to be an effective tool to characterize textures. We apply the oriented filter bank in [47] to three color channels separately. The filter bank has 36 elongated filters at 6 orientations, 3 scales, and 2 phases, 8 center-surround difference of Gaussian filters, and 4 low-pass Gaussian filters. Thus, every pixel has a 144-component filter response vector after filtering. We compute the mean and standard deviation of each component within a superpixel to obtain a 288-component vector.

A normalized histogram is also computed for every color channel of every superpixel. We use 51 bins for each color channel, and the 153-component vector is then concatenated with the 288-component vector. It is reduced to a shorter vector by principal component analysis. This shortened vector, in juxtaposition with the mean and standard deviation of each color channel of the superpixel, forms the texture descriptor for every supervoxel. In our experiments, we have used a descriptor length around 60.

We use AdaBoost [70] with decision trees as our classifier. The above descriptors are fed to our classifier as feature vectors. We limit the maximum depth of each decision tree to 5 to maintain a good generalization capability.

A microscopic image for prostatic cross sections usually contains a large number of gland units. And we typically use multiple training images. To avoid the tremendous work required to prepare the training data by manually segmenting all gland units in the training images, we adopt an interactive training approach [80]. In this approach, the user starts with a few scribbles on one image indicating a few gland regions as well as the background, as shown in Fig. 5.2(b). Superpixels covered by these scribbles serve as the initial positive and negative training data on which the first tentative classifier is trained and applied to the unlabeled data. If the tentative result is not satisfactory, the user inputs a few more scribbles to further guide the training process. In practice, this iterative procedure runs for a few times before the classifier performs sufficiently well on all microscopic images.

5.3.1 Graphcut Based Refinement

As the above classifier labels every patch independently, there may exist sporadic misclassifications in the middle of a gland region as in Fig. 5.2(c). We enhance the spatial coherence of segmentation results by applying a graph cut algorithm. The refined segmentation result is shown in Fig. 5.2(d).

The idea of using graph cut for segmentation [71] is to treat image segmentation as a binary labeling problem. Specifically, the image induces a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$

represents the set of pixels and $\mathcal{E}$ represents the set of edges between every pair of neighboring pixels. Every pixel also has links to two virtual nodes, the source representing the foreground and the sink representing the background. The labeling problem is to assign a unique label l_i for each node $i \in \mathcal{V}$, i.e. $l_i \in \{\text{foreground}(= 1), \text{background}(= 0)\}$, so that the following objective function is minimized.

$$E(L) = \sum_{i \in \mathcal{V}} R(l_i) + \lambda \sum_{(i,j) \in \mathcal{E}} B(l_i, l_j), \tag{5.1}$$

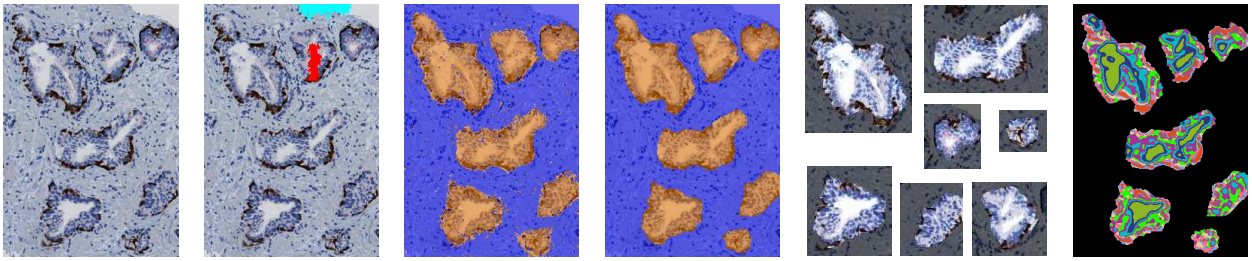
where $R(l_i)$ represents the cost when the label of node i is l_i, and $B(l_i, l_j)$ denotes the cost when the labels of adjacent nodes i and j are l_i and l_j respectively. If we set the weight of edge (i, j) to $B(l_i, l_j)$, the weight of the link between node i and the source to $R(l_i = 1)$ and the weight of the link between node i and the sink to $R(l_i = 0)$, finding an optimal solution of (5.1) is equivalent to finding a cut with minimal cost in the graph. More details can be found in [71] and [40].

We would like to make use of the interactively trained texture classifier in the graph cut algorithm. instead of using a global color distribution to set the source/sink link cost as in [40], we use the signed confidence value (voting result), $\sum_t \alpha_t h_t(p)$, where $\alpha_t h_t(p)$ is the signed label of a pixel p predicted by a base classifier in AdaBoost. Specifically, we set $R(l_i = 0) = \max(0, -\ln \max(0, 0.5 + 0.5 \sum_t \alpha_t h_t(p)))$ and $R(l_i = 1) = \max(0, -\ln \max(0, 0.5 - 0.5 \sum_t \alpha_t h_t(p)))$. We adopt the conventional scheme of assigning edge cost where the color difference between adjacent pixels is used.

5.4 Gland Unit Classification

5.4.1 Gland-level Feature Descriptors

We treat every connected component in the previously segmented foreground as a gland unit (Fig. 5.2(e)). For every detected gland unit, we build a descriptor that characterizes



(a) orig image (b) scribbles (c) initial seg (d) refined seg (e) segmented units (f) textons

Figure 5.2: (a)-(d) shows the gland unit segmentation procedure. (e) shows sub-images of gland units (highlighted regions) segmented from the background (darkened regions). (f) illustrates the distribution of 10 textons. Each color represents a group of pixels associated to a certain texton.

discriminative features of malignant gland units. Such a descriptor will be used for both training and classification. As malignant gland units and benign ones differ in their color distributions, textural and structural features, we incorporate such information when building descriptors. More specifically, we generate a color descriptor, a texton descriptor, and a texton co-occurrence descriptor. They are combined into one single feature vector for every gland unit.

Color distributions are well described by histograms. We compute three normalized histograms for every gland unit, one for each of the three color channels. 51 bins are used for each histogram. Thus, the color descriptor of a gland unit is a 153-component vector.

It is well-known that pixelwise histograms are a type of first-order statistics that is inherently ambiguous. That is, randomly swapping pairs of pixel colors within a gland unit would not affect its histograms. Nevertheless, swapping may destroy the shape and structure of local patterns, such as epithelial cells, making the resulting image unrecognizable as a gland unit. Important visual cues that distinguish a malignant gland unit from a benign one include the percentage of area covered by epithelial cells, the spacing among nearby epithelial cells in the interior of a gland unit as well as along its boundary, and the existence of the lumen. Descriptors more informative about the existence and spatial distribution of certain local patterns would be more useful for our intended classification. Thus, we choose to build descriptors for the frequencies of local patterns as well as descriptors reflecting the

spacing among nearby local patterns.

Since a texture can be considered as a repetitive spatial arrangement of local texture patterns (textons), we start with sorting out typical local patterns by building a texton library for gland units. Since it is possible to reconstruct a local pattern from its responses to a bank of redundant oriented filters, a local pattern can be uniquely represented by its filter responses. We adopt the same oriented filter bank [47] used in Section 5.3, which leads to a 144-component filter response vector at every pixel. Following [47], we cluster these filter response vectors using the K-means algorithm and the resulting cluster centers represent a small set of prototype response vectors, *i.e.* textons. An example of texton distribution can be seen in Fig. 5.2(f).

We form two descriptors based on textons, a texton histogram and a texton co-occurrence descriptor. Every pixel within a gland unit is first associated with a texton whose filter response vector is most similar to the one at the pixel. We mark every pixel with the label of its associated texton. A normalized texton histogram is then computed. The number of bins in the histogram is equal to the number of distinct textons. This histogram holds the information regarding the existence and frequency of typical local patterns.

Inspired by the grey-level co-occurrence matrix [81], we propose another descriptor based on texton co-occurrence frequencies. Recall that every pixel is associated with a texton label, the texton co-occurrence descriptor measures how often a pair of texton labels at a certain distance occur in a gland unit. Within the gland unit, we tabulate the number of occurrences of all possible pairwise texton combinations at specified distances. Each row of the table represents a specific pair of texton labels, and each column corresponds to a certain distance between pairs of texton labels. Thus, each cell in the table records the number of times a specific texton pair, separated by a given distance, appears in the image. The table is then flattened into a single vector. In our experiments where 10 textons are trained, we normally choose 6 distance values ranging from 1 to 24 pixels (2,6,10,15,20,24), resulting in a descriptor vector of length 330. To make descriptors from different-sized gland units

comparable to each other, we normalize them by making the summation of all elements in the vector equal to one. Unlike conventional co-occurrence matrices, our descriptor is rotation-invariant because we do not maintain the orientation of the displacement between a texton pair.

Finally, we concatenate all three normalized descriptors into the same vector and perform PCA on such concatenated vectors to reduce their dimensionality to 80. The vector of PCA coefficients resulting from dimension reduction serves as the final feature vector fed into a classifier.

5.4.2 Training and Classification

Gland units in cross-sectional microscopic images are present to pathologists for labeling. These labels (both malignant and benign ones) as well as their corresponding feature vectors are used for the training of a binary classifier.

Once a classifier has been trained, undiagnosed gland units in new cross-sectional images can be readily segmented and classified. Note that being a flexible framework, any binary classification technique can apply here. In this project, we use support vector machines. We have also experimented with decision tree classifiers enhanced by the AdaBoost algorithm [70], which render satisfactory classification accuracy as well. The discriminative power of the decision trees is greatly improved by the boosting mechanism. As a result, the classifier only requires shallow decision trees of moderate accuracy.

5.5 Experiments

All experiments are carried out on a 3.8GHz Pentium 4 processor. We have obtained 27 microscopic images from which 267 gland units are successfully extracted using the segmentation algorithm described in Section 5.3. Among them, 159 are benign (positive data) and 108 are malignant (negative data). Parameter specifications are given in Section 5.3 and

5.4.1.

We perform a 10-fold cross validation on the dataset and calculate the confidence interval for the accuracy at a 95% confidence level. We experimented with both boosted trees and support vector machines as classifiers. 11 boosted trees of depth 4 are used for the Adaboost algorithm. For SVMs, we have chosen an RBF kernel where parameters C and γ are automatically tuned using the LIBSVM package [82].

Classification results are summarized in the first row of Table 5.1. Descriptors are comprised of color histogram, texton histogram as well as texton co-occurrence. To demonstrate the discriminative power exhibited by our descriptors, we examine other possible feature combinations for building descriptors. In addition to the three features introduced in Section 5.4.1, we have extracted for every gland unit two more features. One is a filter response feature consisting of the mean and standard deviation of per-pixel response within the gland unit. We use the same filter bank as in Section 5.3. The other is a modified grey-level co-occurrence matrix (GLCM), where gray scale is evenly divided into 10 levels, and the orientation information is excluded.

Results of typical feature combinations are given in Table 5.1. As expected, color histogram alone as a descriptor has the least accurate prediction. In combination with filter responses, GLCM, or the first-order texton statistics, it yields better results ranging from 77% to 82%. However, all these combinations have none or incomplete texton statistics, and therefore miss shape and structure information crucial for malignant gland unit discrimination. They are outperformed by our descriptor which includes both first and second order local texton statistics.

5.6 Conclusions

We have introduced effective image segmentation and classification methods for automatic detection of malignant gland units in microscopic images. Both segmentation and classifica-

Feature sets	Classifiers	Accuracy (%)
color histogram + texton histogram + texton co-occurrence	SVM boosted trees	86.0 ± 1.9 85.3 ± 2.2
color histogram + filter response + GLCM	SVM boosted trees	82.1 ± 2.2 82.3 ± 2.3
filter response + GLCM	SVM boosted trees	79.7 ± 2.5 80.2 ± 2.2
color histogram + texton histogram	SVM boosted trees	81.5 ± 1.9 81.0 ± 2.4
color histogram + filter response	SVM boosted trees	79.8 ± 2.3 80.6 ± 1.8
color histogram + GLCM	SVM boosted trees	78.5 ± 1.9 77.6 ± 2.1
color histogram	SVM boosted trees	73.4 ± 2.2 74.6 ± 2.4

Table 5.1: Classification results according to different feature combinations as descriptors.

tion methods are based on carefully designed feature descriptors, including color histograms and texton co-occurrence tables. The framework is designed to be flexible. Techniques for gland unit segmentation, feature extraction, and binary classification can all be tailored or substituted to meet the specific needs of the input data.

Chapter 6

Conclusion

In this paper, we have described several problems in image processing that are handled by hybrid approaches. We have introduced an effective vector-based representation and its associated vectorization algorithm for full-color raster images. Our representation is based on a triangular decomposition of the image plane. The boundaries of a triangular patch in this decomposition are represented using Bézier curves and internal color variations of the patch are approximated using thin-plate splines. Experiments and comparisons have indicated that our representation and associated vectorization algorithm can achieve a more accurate and compact vector-based representation than existing ones. The success of our approach is largely due to the fact that an input image can be treated as a surface mesh embedded in 3D space by taking the color channel as the third dimension. We therefore are able to apply mesh simplification and surface approximation techniques to the original problem of image vectorization.

We have also proposed a novel interactive approach to the problem of texture selection and editing across both spatial and temporal domains. By using a robust classifier in a supervised fashion, our algorithm is able to quickly distinguish the user specified textures or textured objects. Meanwhile, it adopts the idea of active learning and puts the user in a more laid back position where only modest marking and labeling work is required. With its achievements in both solution accuracy and efficiency, our system is a convenient tool for texture selection and subsequent editing. In this project, typical image filtering is used as a preprocessing step, and the active learning methodology is then adopted for the interactive session. Also, by transforming the input image to a connected graph, we are able

to refine the segmentation result using variants of classic graph theory algorithms such as maximum flow. A variant of this approach also proves successful for automatic detection of malignant gland units in cross-sectional microscopic images. Both gland unit segmentation and classification methods are based on carefully designed feature descriptors, including color histograms and texton co-occurrence tables. The framework is designed to be flexible. Techniques for gland unit segmentation, feature extraction, and binary classification can all be tailored or substituted to meet the specific needs of the input data.

References

[1] M. Isenburg and P. Lindstrom, in *Proceedings of Visualization '05* (2005), pp. 231–238.

[2] M. Isenburg, P. Lindstrom, and J. Snoeyink, in *Proceedings of 3rd Symposium on Geometry Processing* (2005), pp. 111–118.

[3] J. Wu and L. Kobbelt, in *Graphics Interface 2003 Proceedings* (2003), pp. 185–192.

[4] M. Isenburg, P. Lindstrom, S. Gumhold, and J. Shewchuk, in *Proceedings of Graphics Interface 2006* (2006), pp. 115–121.

[5] H. T. Vo, S. P. Callahan, P. Lindstrom, V. Pascucci, and C. T. Silva, IEEE Transactions on Visualization and Computer Graphics **13**, 145 (2007).

[6] J. Shewchuk, in *Proceedings of the 11th International Meshing Roundtable* (2002), pp. 115–126.

[7] P. Hansbo, Communications in Numerical Methods in Engineering **11**, 455 (1995).

[8] L. Freitag, P. Knupp, T. Munson, and S. Shontz, in *Proceedings, 11th International Meshing Roundtable* (2002), pp. 29–40.

[9] T. Munson, SIAG/Optimization News and Views **16**, 27 (2005).

[10] L. A. Freitag and P. M. Knupp, in *Proceedings of the 8th International Meshing Roundtable* (1999), pp. 247–258.

[11] L. Freitag, P. Knupp, T. Leurent, and D. Melander, in *Proceedings of the 8th International Conference on Numerical Grid Generation in Computational Field Simulations* (2002), pp. 159–168.

[12] X. Jiao, in *Proceedings, 15th International Meshing Roundtable* (2006).

[13] L. Freitag, M. Jones, and P. Plassmann, SIAM J. Sci. Comput. **20**, 2023 (1999), ISSN 1064-8275.

[14] P. M. Knupp, SIAM Journal on Scientific Computing **23**, 193 (2001).

[15] H.-H. Chang and Y. Hong, Pattern recognition **31**, 1747 (1998).

[16] J. J. Zou and H. Yan, in *CGI '01: Computer Graphics International 2001* (IEEE Computer Society, 2001), pp. 225–231.

[17] X. Hilaire and K. Tombre, IEEE Trans. Pattern Anal. Mach. Intell. **28**, 890 (2006).

[18] D. Nehab and H. Hoppe, ACM Trans. Graph. **27**, Article 135 (2008).

[19] G. Lecot and B. Levy, in *Proceedings of Eurographics Symposium on Rendering* (2006), pp. 349–360.

[20] S. Swaminarayan and L. Prasad, in *AIPR '06: Proceedings of the 35th Applied Imagery and Pattern Recognition Workshop* (IEEE Computer Society, 2006), p. 28.

[21] L. Demaret, N. Dyn, and A. Iske, Signal Processing **86**, 1604 (2006).

[22] B. Price and W. Barrett, The Visual Computer **22**, 661 (2006).

[23] J. Sun, L. Liang, F. Wen, and H.-Y. Shum, ACM Trans. Graph. **26**, Article 11 (2007), ISSN 0730-0301.

[24] Y.-K. Lai, S.-M. Hu, and R. Martin, ACM Trans. Graph. **28**, Article 85 (2009).

[25] A. Orzan, A. Bousseau, H. Winnemöller, P. Barla, J. Thollot, and D. Salesin, ACM Trans. Graph. **27**, Article 92 (2008).

[26] J. Canny, IEEE Trans. Pat. Anal. Mach. Intell. **8**, 679 (1986).

[27] P. D. Kovesi, *MATLAB and Octave functions for computer vision and image processing*, available from: <http://www.csse.uwa.edu.au/∼pk/research/matlabfns/>.

[28] W. E. Lorensen and H. E. Cline, Computer Graphics **21** (1987).

[29] G. Taubin, in *Proc. SIGGRAPH'95* (1995), pp. 351–358.

[30] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, in *Computer Graphics (SIGGRAPH Proceedings)* (1993), pp. 19–26.

[31] M. Garland and P. Heckbert, in *SIGGRAPH 1997* (1997), pp. 209–216.

[32] L. Branets and G. F. Carey, Transactions of the ASME **5** (2005).

[33] R. Fletcher, *Practical Methods of Optimization* (John Wiley & Sons, 1987), 2nd ed.

[34] M. Powell, in *Computational Techniques and Applications* (1995).

[35] S.-Y. Lee, K.-Y. Chwa, and S. Y. Shin, in *SIGGRAPH '95* (ACM, 1995), pp. 439–448.

[36] NVidia, *Nvidia CUDA programming guide 2.0*, http://developer.nvidia.com/object/cuda.html (2008).

[37] A. Patney and J. D. Owens, ACM Trans. Graph. **27**, Article 143 (2008).

[38] T. Lindeberg, Int'l Journal of Computer Vision **30**, 77 (1998).

[39] Y. Li, J. Sun, C.-K. Tang, and H.-Y. Shum, ACM Transaction on Graphics **23**, 303 (2004).

[40] C. Rother, A. Blake, and V. Kolmogorov, ACM Transaction on Graphics **23**, 309 (2004).

[41] Y. Li, J. Sun, and H.-Y. Shum, ACM Transaction on Graphics **24**, 595 (2005).

[42] J. Wang, P. Bhat, A. Colburn, M. Agrawala, and M. Cohen, ACM Transaction on Graphics **24**, 585 (2005).

[43] H. Seung, M. Opper, and H. Sompolinsky, in *5th Annual Workshop on Comput. Learning Theory* (1992), pp. 287–294.

[44] N. Abe and H. Mamitsuka, in *Fifteenth International Conference on Machine Learning* (1998), pp. 1–9.

[45] V. Iyengar, C. Apte, and T. Zhang, in *Sixth ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining* (2000), pp. 92–98.

[46] B. Manjunath and W. Ma, IEEE Transactions on Pattern Analysis and Machine Intelligence **18**, 837 (1996).

[47] T. Leung and J. Malik, Intl. Journal Computer Vision **43**, 29 (2001).

[48] T. Chang and C. Kuo, IEEE Transactions on Image Processing **2**, 429 (1993).

[49] D. Dunn, W. Higgins, and J. Wakeley, IEEE Trans. Pattern Anal. Machine Intell. **16** (1994).

[50] J. Malik, S. Belongie, T. Leung, and J. Shi, Int'l Journal of Computer Vision **43**, 7 (2001).

[51] D. Martin, C. Fowlkes, and J. Malik, in *Neural Information Processing Systems(NIPS)* (2002).

[52] S. Avidan, in *Proceedings of European Conference Computer Vision (ECCV)* (2006).

[53] J. Shotton, J. Winn, C. Rother, and A. Criminisi, in *Proceedings of European Conference Computer Vision (ECCV)* (2006).

[54] E. Mortensen and W. Barrett, in *SIGGRAPH 95 Proceedings* (1995), pp. 191–198.

[55] M. Gleicher, in *SIGGRAPH 95 Proceedings* (1995), pp. 183–190.

[56] A. Agarwala, M. Dontcheva, M. Agrawala, S. Drucker, A. Colburn, B. Curless, D. Salesin, and M. Cohen, ACM Transaction on Graphics **23**, 294 (2004).

[57] A. Protiere and G. Sapiro, IEEE Transactions on Image Processing **16**, 1046 (2007).

[58] L. Gavish, L. Wolf, L. Shapira, and D. Cohen-Or, *Principal-channels for one-sided object cutout*, Tech. Rep., Tel-Aviv University (2007).

[59] J. Wang, in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2007* (2007), pp. 601–604.

[60] A. Levin, D. Lischinski, and Y. Weiss, ACM TOG **23**, 689 (2004).

[61] R. Irony, D. Cohen-Or, and D. Lischinski, in *Eurographics Symposium on Rendering* (2005).

[62] Y. Qu, T.-T. Wong, and P.-A. Heng, ACM TOG **25**, 1214 (2006).

[63] Q. Luan, F. Wen, D. Cohen-Or, L. Liang, Y. Xu, and H. Shum, in *Eurographics Symposium on Rendering* (2007).

[64] Y. Li, E. Adelson, and A. Agarwala, Computer Graphics Forum **27**, 1255 (2008).

[65] S. Brooks and N. Dodgson, in *SIGGRAPH 2002 Proceedings* (2002), pp. 653–656.

[66] K. Bhat, S. Seitz, J. Hodgins, and P. Khosla, ACM Transactions on Graphics **23**, 358 (2004).

[67] Y.-Y. Chuang, D. Goldman, K. Zheng, B. Curless, D. Salesin, and R. Szeliski, ACM Transaction on Graphics **24**, 853 (2005).

[68] P. Felzenszwalb and D. Huttenlocher, Intl. Journal of Computer Vision **59**, 167 (2004).

[69] D. Hoiem, A. Efros, and M. Hebert, ACM Transactions on Graphics **24**, 577 (2005).

[70] R. Schapire, in *MSRI Workshop on Nonlinear Estimation and Classification* (2002).

[71] Y. Boykov and M. Jolly, in *Intl. Conf. Computer Vision* (2001), vol. I, pp. 105–112.

[72] Y.-Y. Chuang, B. Curless, D. Salesin, and R. Szeliski, in *Proceedings of IEEE Conf. Computer Vision and Pattern Recognition* (2001), pp. 264–271.

[73] B. Julesz, Nature(London) **290**, 91 (1981).

[74] D. Pitts, B. Premkumar, A. Houston, R. Badain, and P. Troncosa, in *SPIE Proceedings of Medical Imaging: Image Processing* (1993), vol. 1898, pp. 456–470.

[75] K. Jafari-Khouzani and H. Soltanian-Zadeh, IEEE Transactions on Biomedical Engineering **50**, 697 (2003).

[76] S. Doyle, A. Madabhushi, M. Feldman, and J. Tomaszeweski, in *Medical Image Computing and Computer Assisted Intervention* (2006), vol. 9, pp. 504–511.

[77] A. Tabesh, M. Teverovskiy, H. Pang, V. Kumar, D. Verbel, A. Kotsianti, and O. Saidi, IEEE Transactions on Medical Imaging **26**, 1366 (2007).

[78] P.-W. Huang and C.-H. Lee, IEEE Transactions on Medical Imaging **28**, 1037 (2009).

[79] Y. Zhu, S. Williams, and Z. R., Medical Image Analysis **10**, 178 (2006).

[80] T. Xia, Q. Wu, C. Chen, and Y. Yu, The Visual Computer Journal **26** (2010).

[81] R.Haralick, K. Shanmugam, and I. Dinstein, IEEE Transactions on Systems, Man, and Cybernetics **SMC-3**, 610 (1973).

[82] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines* (2001), software available at `http://www.csie.ntu.edu.tw/ cjlin/libsvm`.